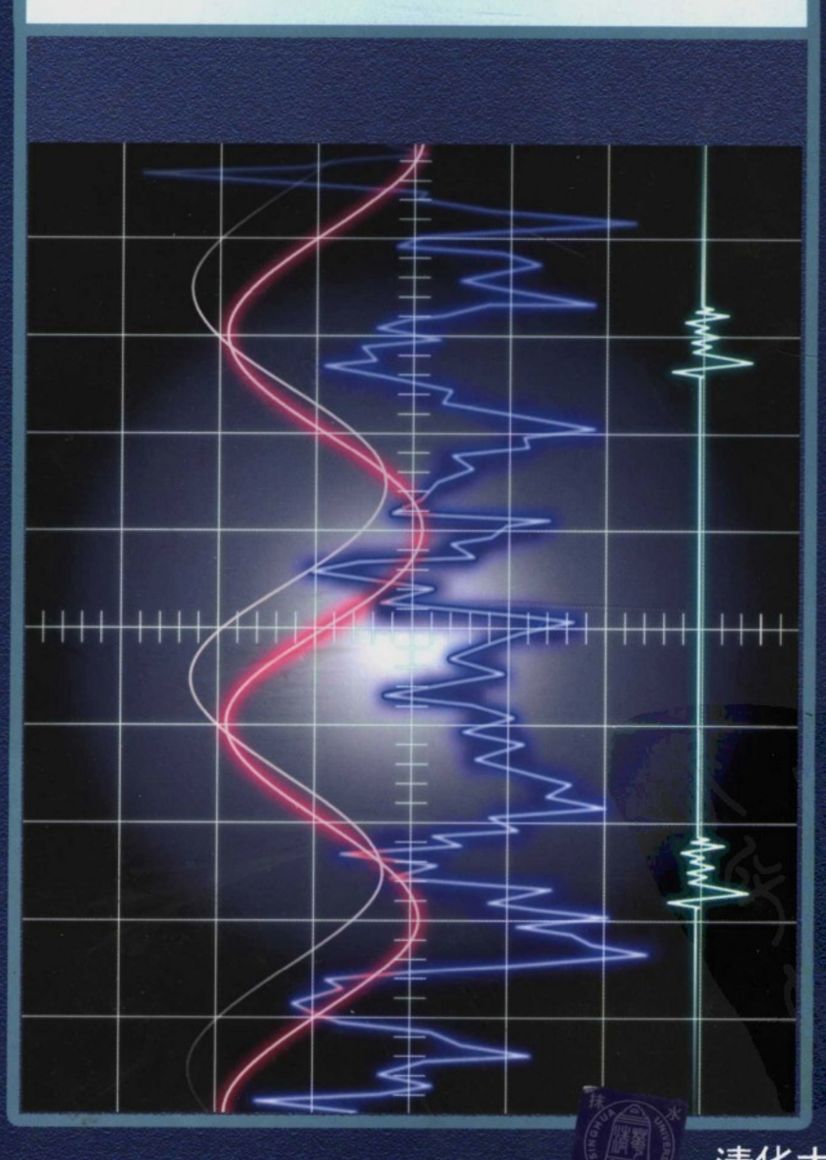
MATLAB 7.0/Simulink 6.0 建模仿真开发与高级工程应用

黄永安　马 路　刘慧敏　编著



清华大学出版社

MATLAB 7.0/Simulink 6.0 建模仿真开发与高级工程应用

本书对当今工程仿真领域应用最为广泛的MATLAB/Simulink软件进行了系统的介绍，采用纵横结合的方法，纵向介绍了Simulink中各模块的功能和操作，横向介绍了Simulink各个工具箱在科学和工程等领域中的应用。

本书特点：

- 采用大量的实例讲解Simulink中各个工具箱的基本使用
- 内容涉及机械、电力、土木、力学、智能科学和虚拟现实等领域
- 本书是对Simulink仿真应用全面系统的介绍，而目前市面上都是针对Simulink某个工具箱的介绍

为了方便读者学习，本书相关内容可登录科研中国(http://SciEi.com)下载。


ISBN 7-302-11796-9
9 787302 117964 >
定价：32.00元

新书查询及技术支持：http://www.wenyuan.com.cn
读者服务邮箱：service@wenyuan.com.cn

MATLAB 7.0/Simulink 6.0 建模仿真开发与高级工程应用

黄永安　马　路　刘慧敏　编著

清华大学出版社
北　京

内 容 简 介

MATLAB/Simulink 是功能强大的仿真软件。本书对整个 Simulink 系统进行了较为全面的介绍。其中包括 Simulink 的使用方法、Simulink 的开发和工程计算问题、Stateflow 原理与使用技巧、SimMechanics 机构系统和 SimPowerSystems 电力系统的应用等内容。

本书内容丰富，涉及多个专业领域，是一本难得的系统的工程书籍，能够帮助读者更好地解决问题，可以作为在校大学生、研究生、教师和科研人员的参考手册，亦可作为广大工程技术人员的参考用书。

图书在版编目(CIP)数据

MATLAB 7.0/Simulink 6.0 建模仿真开发与高级工程应用/黄永安，马路，刘慧敏编著.—北京：清华大学出版社，2005.12
ISBN 7-302-11796-9

Ⅰ.M…　Ⅱ.①黄…②马…③刘…　Ⅲ.计算机辅助计算—软件包，MATLAB 7.0、Simulink 6.0　Ⅳ.TP391.75

中国版本图书馆 CIP 数据核字(2005)第 104528 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社 总 机：010-62770175　　**客户服务**：010-62776969
组稿编辑：刘建龙
文稿编辑：宣　颖
排版人员：朱　康
印 装 者：北京国马印刷厂
发 行 者：新华书店总店北京发行所
开　　本：185×260　**印张**：23　**字数**：546 千字
版　　次：2005 年 12 月第 1 版　2005 年 12 月第1 次印刷
书　　号：ISBN 7-302-11796-9/TP・7674
印　　数：1～4000
定　　价：32.00 元

前　　言

计算机仿真目前已经成为解决工程实际问题的重要手段，MATLAB/Simulink 软件已经成为其中功能最强大的仿真软件之一。而仿真领域的重点是建立模型，即在模型建立以后再设计合理的算法对模型进行计算。Simulink 建模与一般程序建模相比更为直观，操作也更为简单，不必记忆各种参数——命令的用法，只要用鼠标就能够完成非常复杂的工作。Simulink 不但支持线性系统仿真，还支持非线性系统仿真；不但支持连续系统仿真，还支持离散系统甚至混合系统仿真；不但本身功能非常强大，而且还是一个开放性体系，可以自己开发模块来增强 Simulink 自身的功能。对于同一个系统模型，利用 Simulink 可以采用多个不同的采样速率，不但能够实时地显示计算结果，还能够显示模型所表示实物的实际运动形式。

较为完整且系统地介绍 MATLAB/Simulink 强大仿真功能的教材还非常少，多数只是偏重介绍某一方面的应用，或者并没有涉及其中的工具箱。为此，本书详细地介绍了 MATLAB\Simulink 的使用，不但介绍了 Simulink 的基本使用方法、高级应用和开发，而且还介绍了 Simulink 中高级工具箱的使用。

本书的最大特色就是系统地介绍 Simulink，实例丰富，讲解深入浅出。某些比较难以理解的章节，都是伴随工程实例的仿真过程讲解的，使读者能够轻松入门，学以致用。

在欧美的高等院校中，掌握 MATLAB 的应用已经成为大学生、研究生和教师的必备技能，其中 Simulink 更是其中较为深层次的应用。国内的高等院校也开设了 MATLAB 课程，而且网络社区也非常繁荣。在这种情况下，结合本人开设论坛的经验和实际中遇到的问题，以及在学习 MATLAB 的过程中的经验和体会编写了本书，主要介绍 Simulink 的强大仿真功能和使用方法。

本书内容

第 1 章～第 6 章主要介绍 Simulink 的使用方法，如模块的功能、调用、修改和仿真的实现。这是教材的基本内容，主要针对没有 Simulink 基础的读者，使其能够快速入门。

第 7 章～第 10 章介绍 Simulink 的高级使用方法，如 Simulink 的数值计算、子系统封装技术等，并配以工程实例。掌握这一部分就基本上可以进行一般的计算和仿真工程。

第 11 章～第 13 章主要讲解 Simulink 的开发和工程计算问题，如 S-function 的开发、函数的回调、动画输出。

第 14 章～第 20 章系统地讲解了 Stateflow 原理与使用技巧、Simulink Control Design、控制系统仿真、神经网络控制、Real-Time Workshop、SimMechanics 机构系统应用。

本书内容比较专业，能够帮助读者更好地解决使用 Simulink 过程中遇到的问题，可以作为在校大学生、研究生、教师和科研人员以及广大工程技术人员的参考用书。

本书主要由黄永安、马路和刘慧敏负责完成，其中黄永安负责第 7、8、9、12、14、17、18、19 章，马路负责第 1～6 章，刘慧敏负责第 10、11 章，其余各章由教师们共同完成。最后由黄永安和徐斌对整体进行协调，保证各章的连续性和相对的独立性。

如果有任何技术问题，欢迎大家到动力学与控制技术论坛 http://www.dytrol.com 进行交流，相信您能够得到满意的答复。也欢迎 MATLAB 爱好者来这里展现您的能力。

由于编者能力有限且时间仓促，虽然在多个 PC 机上经过反复验算，但书中仍难免有疏漏之处，希望广大读者批评指正。

黄永安



目　　录



第 1 章　Simulink 基础与入门

MATLAB 功能强大，可方便地进行科学与工程计算，大大地减小了计算工作量。而且，MATLAB 所采用的算法都是最新最成熟的算法，并能够与各种程序语言进行融合编程，大大地加快了实际开发的速度。Simulink 是一个针对动力学系统建模、仿真和分析的软件包，可以与 MATLAB 实现无缝结合，能够调用 MATLAB 强大的函数库。

通过本章的基本介绍和实例演示，可使读者对 Simulink 的使用和功能有一个初步的了解，有助于读者消除对 Simulink 的陌生感，加快 Simulink 的上手速度。

本书所有章节的例子都是基于 Microsoft Windows XP 系统、MATLAB7.0 和 Simulink 6.0。

本章主要内容包括：

- Simulink 功能介绍
- 房子热传递模型演示模型
- 质量弹簧系统演示模型
- 演示实例
- 设置 Simulink 显示参数

1.1　Simulink 功能介绍

本书所要介绍的 Simulink 是一种图形化仿真工具包，能够进行动态系统建模、仿真和综合分析，可以处理线性和非线性系统，离散、连续和混合系统，以及单任务和多任务系统，并在同一系统中支持不同的变化速率。

1.1.1　交互式仿真工具

Simulink 具有非常高的开放性，提倡将模型通过框图形式表示出来，或者将已有的模型添加组合到一起，或者将自己创建的模块添加到模型当中。Simulink 具有较高的交互性，允许随意修改模块参数，并且可以直接无缝地使用 MATLAB 的所有分析工具。对最后得到的结果可进行分析，并能够将结果可视化显示。Simulink 的一个意图就是让用户在使用 Simulink 的同时能够感受到建模与仿真的乐趣。通过这个平台，可以激发用户不断地提出问题，对问题进行建模。

Simulink 提供了大量的模块，方便用户快速地建立动态的系统模型，只需用鼠标进行简单的拖放和模块间的连接，就能够建立非常复杂的仿真模型，对模型中的连接数量和规模没有限制。

Simulink 非常实用，应用领域很广，可使用的领域包括航空航天、电子、力学、数学、通信、影视和控制等。世界各地的工程师都在利用它来对实际问题建模，解决问题。

1.1.2　图形化动力学系统建模工具

利用 Simulink 工具包可以不受线性系统模型的限制，能够建立更加真实的非线性系统，如在系统中考虑摩擦力、空气阻力、齿轮滑动等。它会将计算机变成一个建模与分析系统的实验室，特别是对于那些无法做实验的系统，如客机机翼的颤动、生物链系统和货币供给系统等。

对于建模，Simulink 提供了非常方便的图形建模方式(GUI)，通过单击和拖放鼠标搭建框图来完成仿真模型的建立。通过 Simulink 提供的窗口，搭建框图就如同用铅笔在白纸上画图一样方便快捷。与以前将微分方程写成某种语言或程序相比，这无疑是一个创举。Simulink 包括非常全面的模块库及工具箱。模块库包括 sinks、sources、linear 和 nonlinear components 以及 connectors。工具箱包括 Real Time Workshop、SimMechanics、Stateflow、Simulink Control Design 和 Virtual Reality Toolbox 等。而且 Simulink 是非常开放性的程序包，用户可以自己方便地建造自己的模块库，定制满足特殊功能的模块或者模块组，后面相关章节有详细的介绍。

模块都是分等级的，所以可以采用从上到下或者从下到上的顺序建模。可以从起点开始建模，然后依次将其连接，最后按顺序双击所要使用的模块进行参数设置。这种建模方法思路清晰，对各模块的相互作用和组织形式一目了然。

在建立好模型之后和运行仿真之前，必须对模型进行参数设置。仿真所需要的模型参数可以通过 MATLAB 命令窗口或者 Simulink 菜单来进行设置。这两种方法各有千秋，前者适合批处理多个仿真，而后者直观方便。设置模块参数可以双击相应模块，在弹出对话框中进行参数设置。仿真完成后，使用 Scope 或 XY Graph 等模块来显示结果，还可以直接将结果输出到 MATLAB 的工作空间，然后利用各种图像处理函数来处理结果。Simulink 除了能够将数据导出到 MATLAB 工作空间中，还可以将 MATLAB 工作空间中的数据导入到 Simulink 模型中。

模型分析工具包括线性化和精简工具，这些都可在 MATLAB 的命令中直接实现。由于 MATLAB 和 Simulink 是集成的，所以既可以在 MATLAB 命令空间，也可以在 Simulink 平台下对模型进行仿真、分析和修正。MATLAB 主要是以键盘输入命令的形式调试运行模型，而 Simulink 主要是通过鼠标设置参数对话框来调试运行模型的。

1.1.3　Simulink 的扩展功能

Simulink 是一个开放式结构体系，允许用户自己开发各种功能的模块，无限制地添加到 Simulink 环境中，以满足不同任务的具体要求。

可以采用以下方式来增强 Simulink 的模块功能：

- 采用 MATLAB 的 M 文件、Fortran 以及 C 代码生成自定义模块。
- 利用 Simulink 本身来建立子系统，封装为一个自定义模块。
- 将 Simulink 与开发好的 S 函数无间隙连接起来，完成复杂的功能。
- 将原有的 Fortran 和 C 代码连接起来。
- 其他大型工程软件衔接的接口，如 Adams 结构仿真软件，可以实现利用

Simulink 对 Adams 中结构的控制。

- 有很多第三方开发的工具箱。
- 很多软件都留有与 MATLAB 方便连接的接口，如 Femlab、Labview 等，使 Simulink 能够非常方便地利用这些软件中的信息以及被这些软件所调用。

1.1.4　Simulink 专用模块库与相关产品

MathWorks 公司针对 Simulink 自己开发或收购了一系列的产品来加强 Simulink 的功能，使得 Simulink 能够满足不同行业不同任务的要求。这些产品包括 Neural Network Blockset、SimPowerSystems、Real Time Workshop、SimMechanics、Stateflow 以及 Virtual Reality Toolbox 等，具体可见：http://www.mathworks.com/products/simulink/related.html。

为丰富 Simulink 建模系统，MathWorks 公司开发了许多有特殊功能的模块程序包，如 Fixed-Point Blockset 和 Communication Blockset 等。利用这些功能强大的程序包，使得用户能够非常方便地建立模型或者完成系统分析。除此之外，还能够将建立的 Simulink 模型生成代码，以便实时控制等。

常用的 Simulink Blockset 有：

- CDMA Reference Blockset (CDMA 通信系统设计与分析)
- Communication Blockset(通信系统工具箱)
- Dials & Gauges Blockset(交互式图形和控制面板设计工具箱)
- DSP Blockset(数字信号处理工具箱)
- Fixed-Point Blockset(定点运算控制系统工具箱)
- Motorla DSP Developer's Kit(Motorola DSP 开发工具)
- Nonlinear Control Design Blockset(非线性控制设计工具箱)
- SimPowerSystems(电力电动工具箱)
- TI DSP Developer's Kit(TI DSP 开发工具)
- SimMechanics(机构仿真)
- Neural Network Blockset(神经网络)
- Stateflow(流程控制)
- Real Time Workshop(实时系统)

1.2　房屋热传递演示模型

Simulink 帮助中提供了一些有趣、复杂、实用的演示模型，在此列举一个关于房屋热力学问题的系统，简单了解一下 Simulink 的运行过程及其功能。

1.2.1　运行演示模型

模型的创建与设置将会在以后的章节中详细介绍，运行模型的操作步骤如下。

(1) 启动 MATLAB。

(2) 在 MATLAB 命令窗口输入：

```
>> thermo
```

直接打开演示程序模型窗口，这个命令会启动 Simulink，打开模型如图 1.1 所示。

(3) 在第(2)步中，打开模型的同时会弹出 Thermo Plots 的 Scope 窗口。如果在弹出时关闭了，可以通过双击图 1.1 中最右侧的 Thermo Plots 模块重新打开。

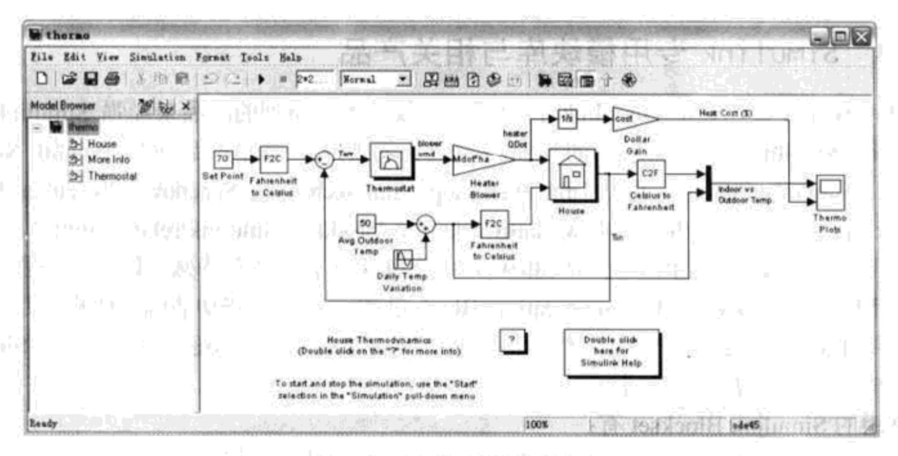


图 1.1 房屋热传递演示模型

(4) 进行仿真。

可通过 4 种方式运行仿真：

- 在 Simulink 中选择 Simulation | Start 命令；
- 按 Ctrl+T 快捷键；
- 单击图标 ▶；
- 在 MATLAB 命令窗口输入：

```
>>sim('thermo.mdl')
```

(5) 运行后，Simulink 中的结果如图 1.2 所示。

(6) 如果要暂停或结束仿真，有两种方法：

- 在 Simulink 中选择 Simulation | Stop 命令；
- 单击图标 ❚❚ 表示暂停，单击图标 ■ 表示结束。

(7) 程序运行完毕后，可在 Simulink 中选择 File|Close 命令来关闭。

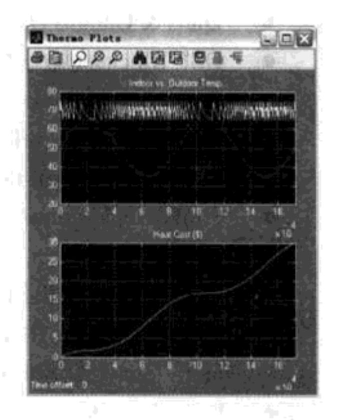


图 1.2 仿真结果

1.2.2 演示模型描述

模型是对房屋热力系统进行的建模仿真，温度调节装置设定为 70° F，但受到室外温度的影响，室外温度为 50+15sin(2×pi/(24×3600)×t)° F，仿真结果是房屋内一天温度变化。

为了简化模型，对模型有一个清晰的认识，本系统采用了 3 个子系统，每个子系统都由若干模块组合而成。这 3 个子系统分别为：

- Thermostat 子系统
- House 子系统

- 3 个温度转换子系统

首先将室内和室外的温度输入到 House 子系统，然后更新室内的温度。双击 House 子系统可得如图 1.3 所示的模型。

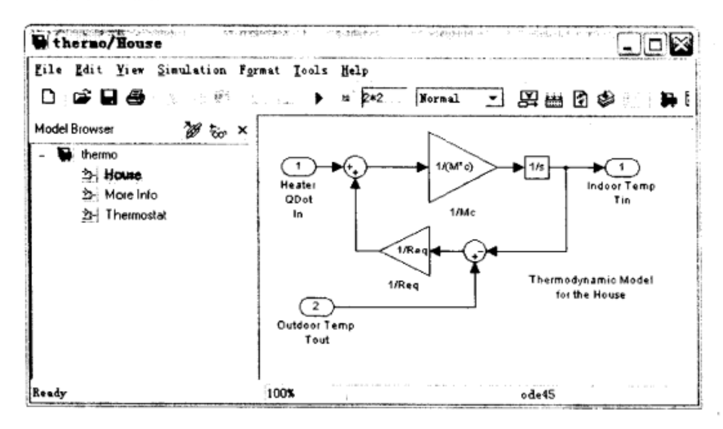


图 1.3　House 子系统模型

Thermostat 子系统是对温度调节器进行建模，设置开与关。双击 Thermostat 子系统可得到如图 1.4 所示模型。室内室外的温度都必须从华氏度(°F)转换到摄氏度(℃)，转换公式：C=5×(F−32)/9。在图 1.1 中双击 Celsius to Fahrenheit 模块会弹出如图 1.5 所示的对话框。图 1.5 是被精细封装的子系统，在后面的章节会讲解如何创建封装子系统，此子系统详细构造如图 1.6 所示。

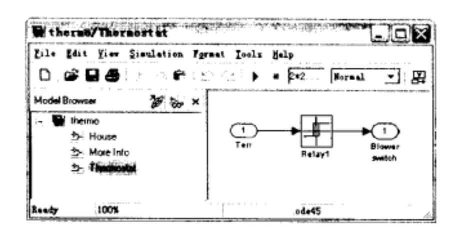


图 1.4　Thermostat 子系统模型

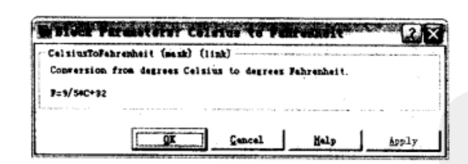


图 1.5　温度转换精细封装子系统

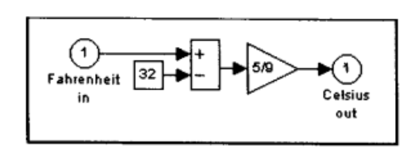


图 1.6　温度转换子系统构造

1.3 质量弹簧系统演示模型

Simulink 已经广泛应用到科研的各个方面，下面就是一个典型的双质量弹簧系统，在光滑平面上受一个周期作用力情况下的运动状态。在左边的质量块上作用一个周期激励，在此模型中使用了状态判断和 LQR 控制。此模型用一个 S 函数编写的程序来显示系统的运动动态图形。

模型中包含了封装子系统，如图 1.7 中所示的 Animation function 模块，可以双击相应的模块，打开对话框或子系统窗口。再次双击模块 Animation function，可以看到如图 1.8 所示对话框，此对话框将设置用动画来显示仿真结果，其中 crtanim2 就是显示动画的 S 函数名称。仿真结果的动画显示如图 1.9 所示。

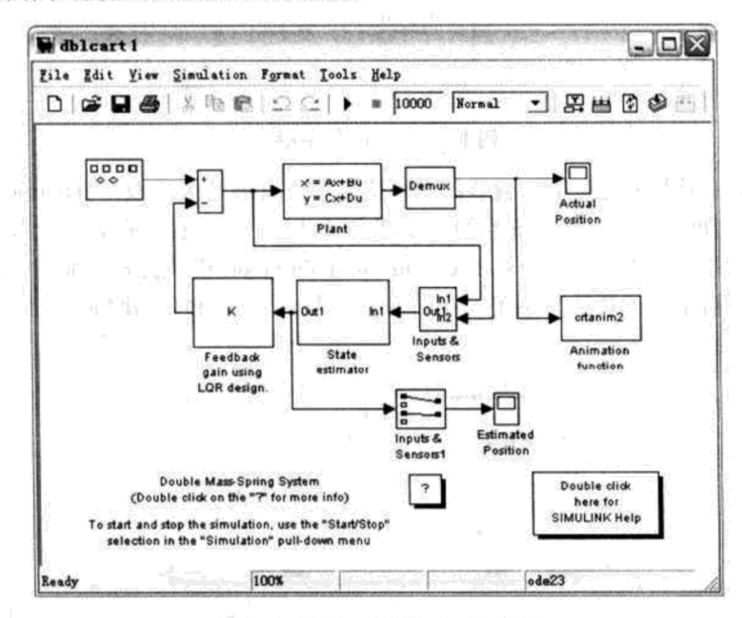


图 1.7 双质量弹簧系统演示模型

Block Parameters: Animation function

S-Function

User-definable block. Blocks may be written in C, Fortran or Ada and must conform to S-function standards. t,x,u and flag are automatically passed to the S-function by Simulink. "Extra" parameters may be specified in the 'S-function parameters' field.

Parameters

S-Function Name: crtanim2 Edit

S-Function Parameters: 0.1

OK Cancel Help Apply

图 1.8 Animation function 参数设置对话框

双击 Actual Position 模块，可以得到质量块的时程分析曲线图，如图 1.10 所示。Feedback gain using LQR design 模块是一个状态反馈增益。

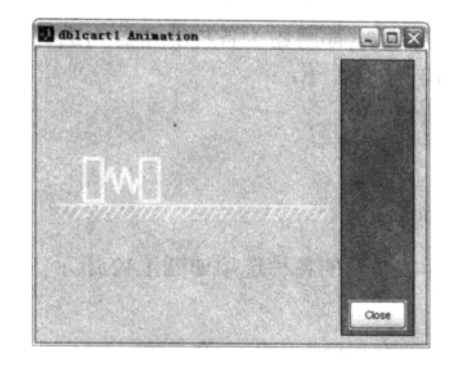


图 1.9　仿真结果动画显示窗口

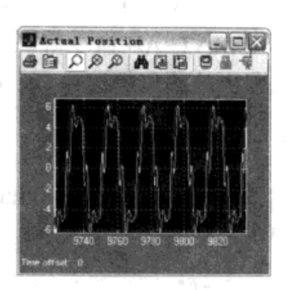


图 1.10　质量块实际位置时程曲线

双击 Estimated Position 模块可看出系统 4 个状态中的两个主要取决于 Inputs&Sensors1 模块参数设置，双击此模块，可看到如图 1.11 所示对话框，设置参数如下：

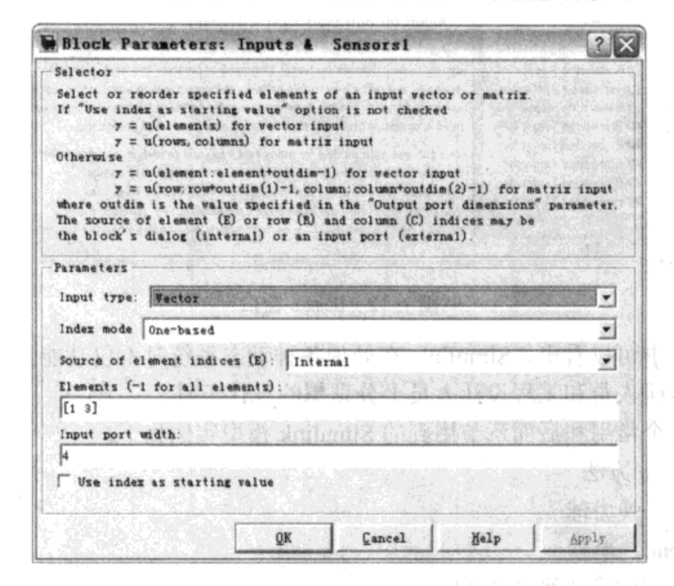


图 1.11　Inputs&Sensors1 模块的参数设置对话框

- Elements 文本框设置所要选取的状态，此模型设置选取的状态为[1 3]。
- Input port width 文本框设置模块的输入状态数量，模型中模块输入的维数为 4。

1.4　更多演示实例

为了帮助用户了解 MATLAB 和 Simulink，MATLAB 除了提供非常详细的说明外，还配备了大量的实例，这些例子有助于提高对 Simulink 的认识与兴趣。

1.4.1 Simulink 模型演示实例

除了 1.2 节和 1.3 节演示之外，下面还有很多演示都阐述了各种非常有用的建模方法。可通过 MATLAB 命令窗口观看这些演示，操作步骤如下：

(1) 单击 MATLAB 命令窗口左下角的 Start 按钮。

(2) 从弹出的菜单中选择 Demos 命令。

说明： 或直接在 MATLAB 窗口中输入 Demos。

(3) 从弹出的窗口中单击 Simulink 前的⊞图标，展开的各种选项如图 1.12 所示。

(4) 单击相应的选项进入不同的演示。

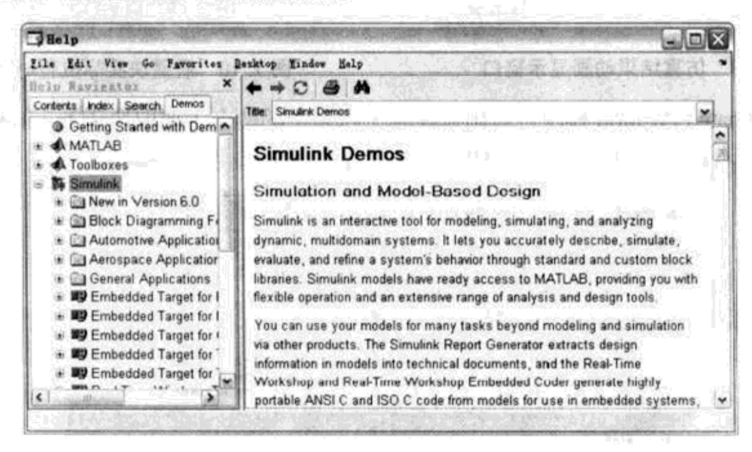


图 1.12 Demos 窗口

从这些例子中可以看出，Simulink 在处理各种动态系统时不仅功能强大，而且十分简单、方便，是科研人员和工程设计人员十分理想的设计工具。

在此提供几个常用和后面章节用到的 Simulink 模型实例如下：

- vdp：vdp 方法
- bounce：弹力球
- simppend：单摆
- onecart：质量弹簧系统(单质点)
- dblcart1：质量弹簧系统(双质点)
- dblpend1：双摆系统
- dblpend2：双摆系统
- penddemo：倒立摆系统

1.4.2 MATLAB 中有趣的演示实例

MATLAB 中也同样有许多有意思的实例，为提高读者对 MATLAB 和 Simulink 的兴趣，特举部分实例以供参考。具体如下，运行的时候只要将“：”前面的代码复制到 MATLAB 中就可以了，随之会出现各种各样的演示实例，对初学者的帮助很大。

- 平面与立体绘图
  graf 2d：XY 平面绘图(火柴棒)
  graf 2d2：XYZ 立体绘图(切片)
  hndlgraf：平面显示线型处理窗口及命令演示
  hndlaxis：平面显示处理窗口及命令演示
  graf3d：立体显示处理窗口及命令演示
- 复杂函数的三维绘图
  cplxdemo：复杂的 XYZ 立体图形
- 等高线绘制
  quivdemo：等高线箭头显示
- 动画
  lorenz：Lorenz 吸引子动画显示
- 电影
  vibes：L 形薄膜振动
- Fourier 变换
  sshow sunspots：太阳黑点数据的傅里叶分析
  fftdemo：分析噪声序列中两组数据的相关度
- 数据拟合
  sshow fitdemo：显示非线性数据拟合过程
  census：预测世界人口
  spline2d：样条拟合
- 稀疏矩阵
  sshow sparsity：降阶
- 游戏
  xpbombs：仿 Windows 系统自节的扫雷游戏
  life：生命发展游戏
- 三维效果图
  klein1：肤色三维效果图
  tori4：4 个首尾相接的圆环
  spharm2：球形和声
  cruller：饼状物
  xpklein：Klein 瓶
  modes：L 形薄膜的 12 种模态
  logo：MATLAB 的 Logo
  xpquad：不同比例的巴尔体超四方体
  truss：二维桁架的 12 模态
  travel：旅行商问题动画演示
  wrldtrv：在地球仪上演示两地间的飞行路线
  makevase：通过单击鼠标来制作花瓶

xpsound：声音样本分析
funfuns：综合了找零点、最小化和单输入函数积分功能
sshow e2pi：e^pi 或 pi^e
quake：地震波可视化
penny：便士可视化
imageext：改变图像的映射颜色
earthmap：地球仪

- 优化工具箱

bandem：香蕉最优化展示
sshow filtdem：滤波效果演示
sshow filtdem2：滤波设计演示
cztdemo：FFT 和 CZT(两种不同类型)的 Z-变换算法
phone：演示电话通声音的时间与频率的关系
sigdemo1：离散信号的时频图，可用鼠标设置
sigdemo2：连续信号的时频图，可用鼠标设置
filtdemo：低通滤波器的交互式设计
moddemo：声音信号的调制
sosdemo：数字滤波器的切片图

- 神经网络工具箱

neural：神经网络模块组
firdemo：二维 FIR 滤波器
nlfdemo：非线性滤波器
dctdemo：DCT 演示
mlpdm1：利用多层感知器神经网络拟合曲线动画
mlpdm2：利用多层感知器神经网络进行 XOR 问题运算

- 模糊逻辑工具箱

invkine：运动逆问题
juggler：跳球戏法
fcmdemo：FCM
slcp：类似倒立摆动画
slcp1：类似倒立摆动画
slcpp1：类似倒立摆动画，有两个摆，一个可以变化
sltbu：卡车支援
slbb：类似于翘翘板

1.5 设置 Simulink 显示参数

设置 Simulink 的显示参数，包括文字的字体及大小、子系统的显示窗口、矢量注明和是否显示模型回调等功能，而非设置 Simulink 模型中的求解器、误差、步长和时间区

间等，这些具体针对每一个模型的设置必须到 Simulink 窗口中设置，具体可以选择 Simulation | Configuration Parameters 命令，从弹出的菜单中进行设置，内容非常多，在此不赘述。

1. Simulink Preferences 参数

MATLAB 中的 Preferences 对话框允许修改 MATLAB 的某些默认参数，包括 Simulink 中的显示参数。这些参数包括文字的字体及大小、子系统的显示窗口、矢量注明和是否显示模型回调等的功能。选择 File|Preferences...命令，弹出 Preferences 对话框，选择其中的 Simulink 目录，如图 1.13 所示，参数详细说明如下。

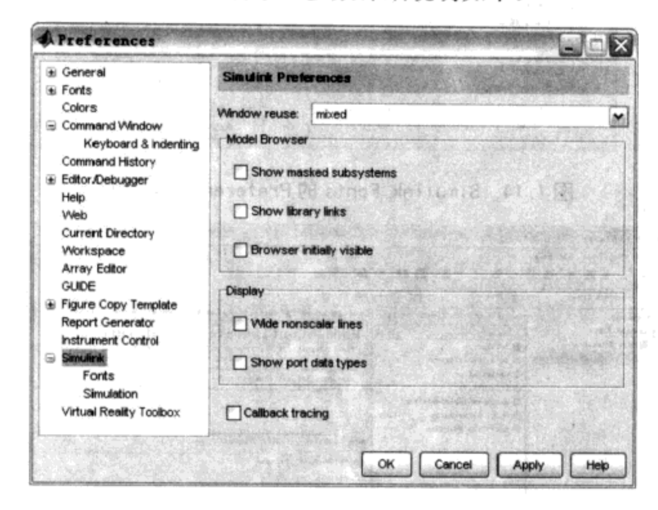


图 1.13 Simulink 的 Preferences 对话框

- Window reuse：确定打开子系统的显示方式，即确定是在一个新的 Simulink 界面打开，还是在原有的界面打开。
- Model Browser：确定在打开模型窗口时是否显示 Simulink Library Browser，是否显示封装子系统，是否显示来源于子系统的模块。
- Display：确定是否用粗线显示矢量，是否显示端口数据类型。
- Callback tracing：确定是否显示模型回调。

2. Simulink Fonts 参数

在如图 1.13 所示的 Preferences 对话框中选择 Simulink 项下的 Font 命令，得到如图 1.14 所示的对话框。在此对话框中，可以设置模块、线型和解释的说明性文字，第一个下拉列表设置字体，第二个列表设置加粗和斜体，最后一个列表设置字体的大小。

3. Simulink Simulation 参数

在图 1.14 所示的 Preferences 对话框中选择 Simulink 项下的 Simulation 选项，会打开相应的对话框，单击 Launch model explorer 按钮，弹出如图 1.15 所示的窗口。

在图 1.15 所示窗口中可以设置 Solver，Data Import/Export，Optimization，Diagnostics，Hardware Implementation，Model Referencing，Real-Time Workshop 等选项的

默认参数，这些参数在以后的章节中将会说明。

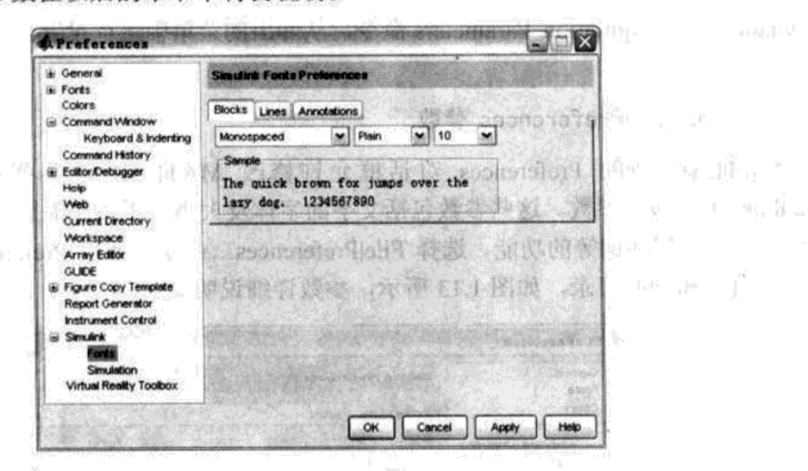


图 1.14 Simulink Fonts 的 Preferences 对话框

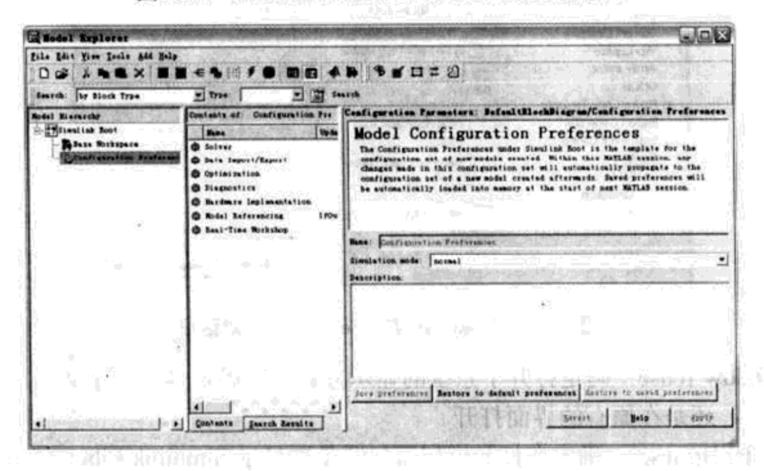


图 1.15 Model Configuration Preferences 参数窗口

第 2 章　Simulink 建模方法

第 1 章中简单介绍了 Simulink 的建模过程以及系统自带的几个实例演示。本章将详细介绍整个模型创建的全过程，包括模块操作、连线操作、编辑信号线及标注模型。同时介绍打印模型，包括直接打印机打印和嵌入 Word 文件中。通过本章的讲解，读者应该能够利用 Simulink 建立一个简单的模型。

本章主要内容包括：

- 打开模型
- 模块操作
- 模块连线操作
- 模型说明
- 模型打印
- 模型文件

2.1 打开模型

Simulink 主要有两种方法来打开模型：

- 直接在 MATLAB 中选择 File | Open 命令，然后按照 Windows 的常规操作进行。
- 直接在 MATLAB 命令窗口中输入模型名，然后 MATLAB 就会自动在工作目录中搜索。例如，在当前工作目录中存在模型文件为 vdp.mdl，只需要在命令窗口中输入模型名：

```
>>vdp
```

2.2 模块操作

本节详细介绍了调整模块大小、旋转模块、复制模块以及对模块进行命名等操作，利用的是模块的建模基础。熟练掌握模块的操作方法，可以使以后的建模过程得心应手。

2.2.1 调整模块大小

通过调整一个模块大小，能够直接清晰地看到模型的参数，提高模型可读性。有些模块，如 Gain 增益模块等，当参数位数较小时，可以直接显示；当参数位数较大时，则以字母代替，此时可适当地扩大模块的大小，使之显示所设置的参数。调整模块的操作步骤如下：

(1) 新建一个模型窗口，并命名为 model02to01.mdl。

(2) 选择 Sources 中的 Constant 模块库，如图 2.1 所示，并将其拖动到模型窗口，双击此模块，并设置 Constant value 文本框中的值为 9999.9999，如图 2.2 所示，由于常数

9999.9999 在图标中不能显示，只显示为“-C-”，如图 2.3 所示。

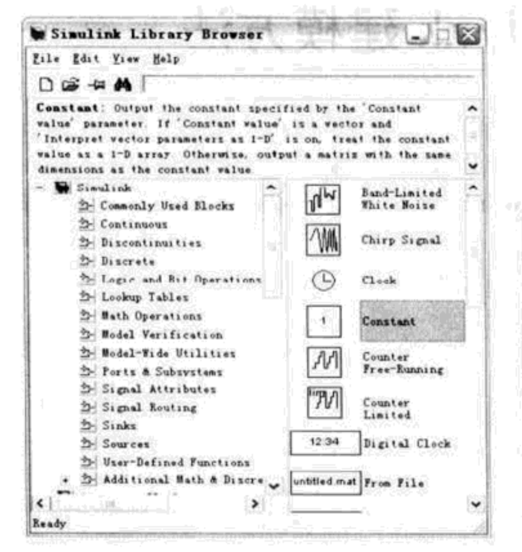


图 2.1　资源模块库

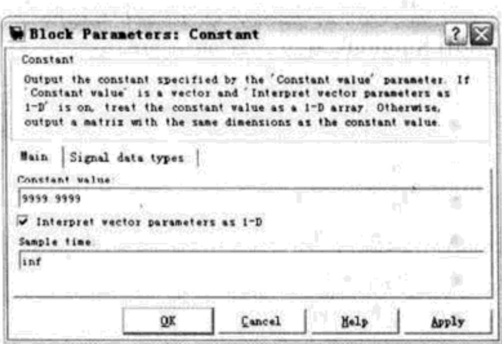


图 2.2　常数模块对话框

(3) 为了能够显示常数，可以扩大模块，单击 Constant 模块，然后用鼠标指针放在四个黑方块上，此时鼠标指针会改变形状，然后拖动鼠标，最后得到的模块形状如图 2.4 所示。

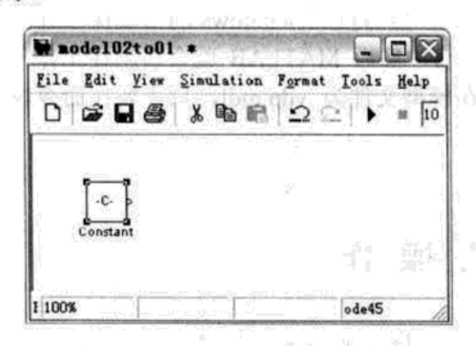


图 2.3　未调整大小的常数模块

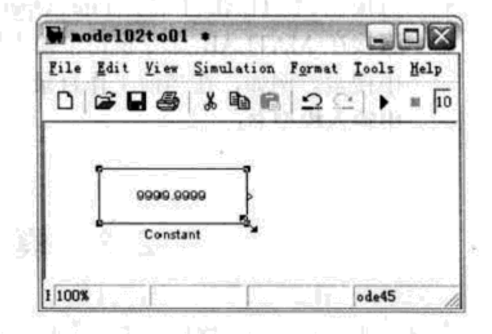


图 2.4　调整大小后的常数模块

2.2.2　模块旋转

进行模块旋转操作，主要有两种方法，可以借用模型 model02to01.md 进行操作。这两种方法分别介绍如下：

- 单击需要旋转的模块，然后选择 Format | Rotate block 命令，模块如图 2.5 所示；
- 单击需要旋转的模块，并右击此模块，会弹出快捷菜单，选择 Format | Rotate block 命令。

Rotate block 是顺时针旋转 90°，Flip block 则是旋转 180°，用法同上。在图 2.5 的基础上进行操作，得到如图 2.6 所示的结果，保存模型文件。

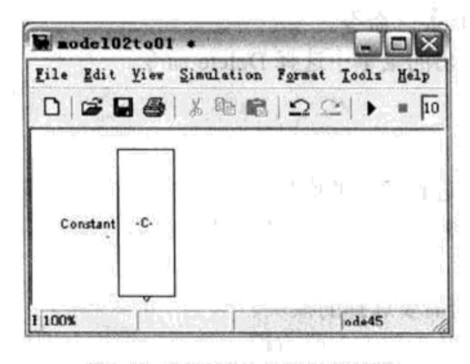


图 2.5 旋转 90° 后的常数模块

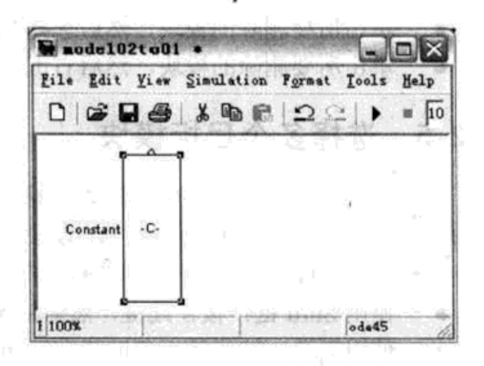


图 2.6 旋转 180° 后的常数模块

2.2.3 模块复制

在建模过程中，经常遇到大量功能重复和设置相同的模块，如果每个都从模块库中拖过来，然后进行参数设置，操作显得非常麻烦、费时，而且容易出错。为了避免这种情况发生，可以直接复制设置好的模块。

下面介绍 4 种方法来复制内部模块，最后得到如图 2.7 所示结果，保存模型文件名为 model02to02.mdl。

- 单击所要复制的模块，然后选择 Edit | Copy 命令，最后选择 Edit | Paste 命令。
- 单击所要复制的模块，然后按 Ctrl+C 组合键，最后按 Ctrl+V 组合键。
- 单击所要复制的模块，按住 Ctrl 键，然后用鼠标拖动要复制的模块。
- 按住鼠标的右键拖动要复制的模块，这种方法最为方便，推荐使用。

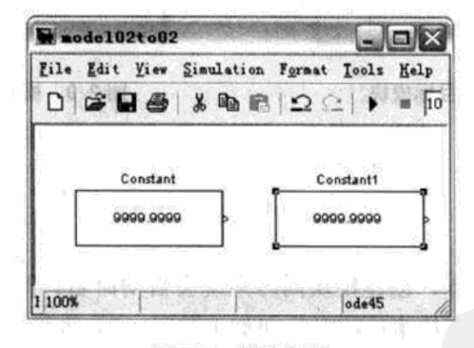


图 2.7 模块复制

2.2.4 模块删除

当模型中出现了多余的模块，即使不删除，Simulink 也能照样运行，并不会因此而影响仿真结果。但是多余的模块会降低模型的可读性，并会在 MATLAB 命令窗口中出现大量的警告信息，这十分不利于调试程序。

删除模块通常有下面 3 种方法：

- 单击所要删除的模块，然后按 Delete 键，比较方便，推荐使用。

- 单击所要删除的模块，然后选择 Edit | Delete 命令。
- 单击所要删除的模块，然后右击，从弹出的菜单中选择 Delete 命令。

2.2.5 选择多个目标模块

在建模过程中，有时候往往需要对多个模块进行同样的操作，如复制、旋转、删除、移动等。在进行这些操作之前，可以通过一次性选择多个目标模块来加快操作的速度。

选择多个目标模块主要有以下两种方法：

- 使用 Shift 键：按住此键，然后依次单击需要选择的模块。
- 使用框选：按住鼠标左键或右键均可，从任何方向画方框，使画出来的方框框住要选择的模块，如图 2.8 所示，文件名为 model02to03.mdl。

框选后的模块如图 2.9 所示。

说明： 两种方法中，前者适合于多个零散模块的选择，后者适合于整篇模块的选择。按住 Shift 键可以选择模块，同样也可以取消已选择的模块，用户可以灵活运用，以加快建模的速度。

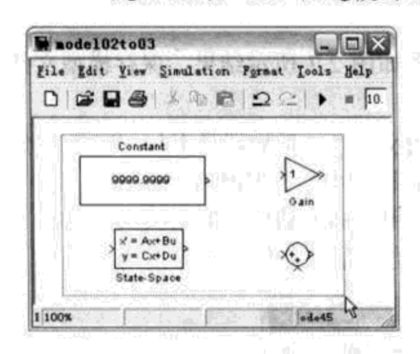


图 2.8 框选多个目标模块

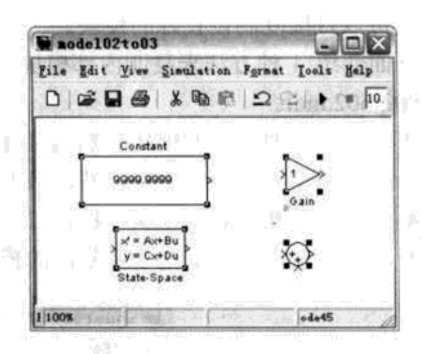


图 2.9 框选多个目标模块后

2.2.6 标签设置

1. 修改模块的标签

每个模块都有一个标签，创建模块同时系统会自动命名。例如图 2.10 中，如果有多个相同模块，系统会自动在原来模块名后面加上数字，如 Gain1。如果有多个，将会依次为 Gain2，Gain3 等。模块标签不可同名，这不同于连线标签。但很多情况下，如果希望修改这个系统特定标签，以提高系统或模块的可读性。可以通过修改模块标签来达到这个目的。

修改模块标签的操作方法为：在所要修改的标签上面单击，标签则呈现可编辑状态。如图 2.10 所示，输入想要的标签，在此输入“SciEi.com”，设置完成在空白域单击，结果如图 2.11 所示，保存模型文件为 model02to04.mdl。

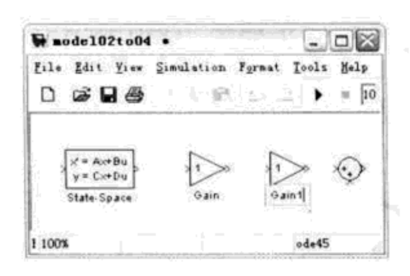


图 2.10　可编辑状态标签

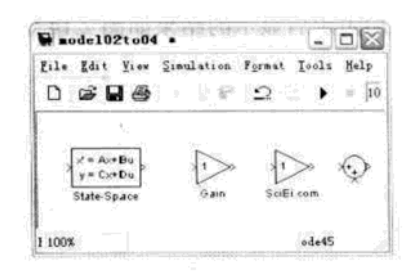


图 2.11　标签编辑完成后

2. 修改标签位置

修改标签位置主要有两种操作方法，具体如下：

- 单击所要编辑的模块，然后选择 Format | Flip name 命令。
- 单击所要编辑的模块，然后右击，从弹出菜单中选择 Format | Flip name 命令。

改变 SciEi.com 模块标签位置后的结果如图 2.12 所示。

3. 隐藏标签

隐藏标签主要有两种操作方法，具体如下：

- 单击所要编辑的模块，然后选择 Format | Hide name 命令；
- 右击所要编辑的模块，从弹出菜单中选择 Format | Hide name 命令。

隐藏 SciEi.com 模块和 Gain 模块的标签，结果如图 2.13 所示。

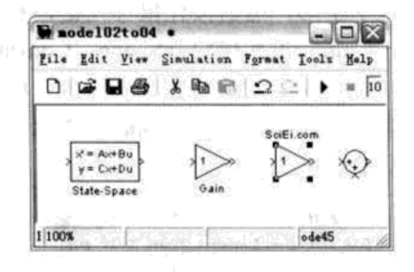


图 2.12　修改标签的位置

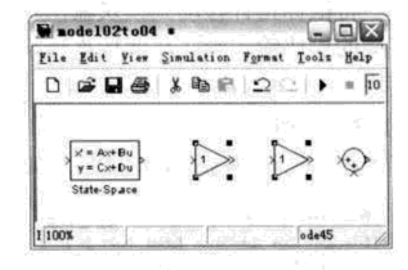


图 2.13　隐藏标签

4. 显示标签

显示标签与隐藏标签作用相反，操作方式类似，主要有两种操作方法，具体如下：

- 单击所要编辑的模块，然后选择 Format | Show Name 命令。
- 单击所要编辑的模块，然后右击，从弹出菜单中选择 Format | Show Name 命令。

2.2.7　增加模块阴影

为提高系统的可读性，或者突出模型中的重点模块等，可以通过为模块增加阴影来凸现模块，能够增强视觉效果，有助于理解模型系统。主要有两种操作方法，具体如下：

- 单击所要编辑的模块，然后选择 Format | Show Drop Shadow 命令。
- 右击所要编辑模块，从弹出菜单中选择 Format | Show Drop Shadow 命令。

增加阴影后显示的结果如图 2.14 所示。

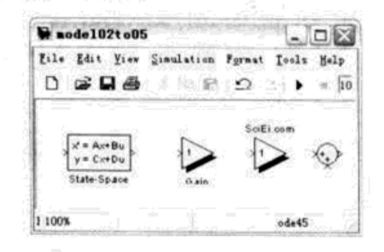


图 2.14 增加阴影效果

2.3 模块连线操作

模型中有模块，还必须有连线将模块联系起来才能够变成一个有机整体。模块和连线是模型的骨架，模块和模型的参数设置是模型的灵魂。下面就对连线的几个基本操作进行讲解。

1. 绘制连线

绘制连线的操作步骤如下：

(1) 新建模型窗口，保存文件名为 model02to06.mdl。向窗口中添加相应的模块，在此不要求实现一个运行的模型，只需任意拖动两个模块到模型窗口中，如图 2.15 所示。

(2) 将鼠标指针移动到模块输出端，鼠标指针呈十字形，然后按住鼠标左键，移动到所要连接的模块输入端，在此依次连接 Sine Wave—Gain—求和模块—Scope，Constant—求和模块。

(3) 绘制模块 Gain 输入端的连线，将鼠标指针移动到 Constant—Scope 连线上，按住鼠标右键，并拖动到 Scope 输入端，如图 2.16 所示。

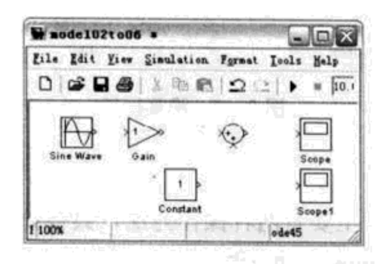


图 2.15 添加模块

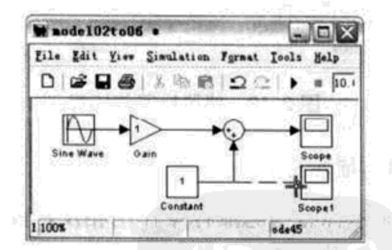


图 2.16 绘制连线

2. 连线移动

在复杂的模型中，由于有许多连线，而且连线之间往往容易交叉，这就降低了模型的可读性，因此有必要拖放连线。连线移动的操作步骤如下：

(1) 单击希望移动的连线。

(2) 将鼠标指针移到连线上，鼠标指针形状会变为移动图标(十字形)，按住鼠标左键并拖动鼠标，前后对比如图 2.17 所示。

3. 节点移动

此操作类似于连线移动，只是将鼠标指针放在连线的转角处。此时鼠标指针的形状会变成圆形，再拖放节点到期望的地方即可，如图 2.18 所示。

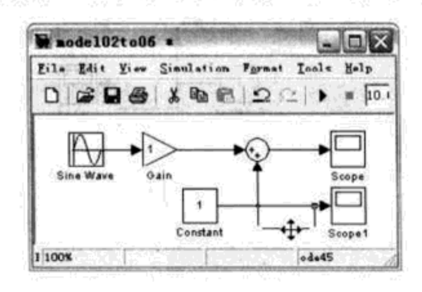


图 2.17 连线移动前

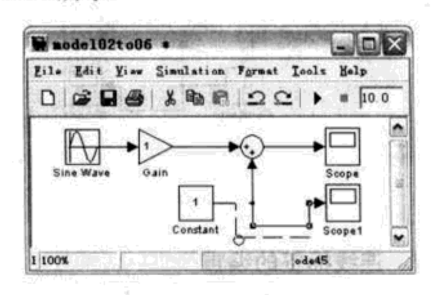


图 2.18 节点移动后

4. 连线删除

删除连线和删除模块一样，有 3 种方法：

- 单击所要删除的连线，然后按 Delete 键。
- 单击所要删除的连线，然后选择 Edit | Delete 命令。
- 单击所要删除的连线，然后右击，在弹出菜单中选择 Delete 命令。

5. 连线分割

在模型 model02to07.mdl 文件窗口中选择要编辑的连线，按住 Shift 键，在要分割的地方单击，其形状就会变成圆形，而连线也就在此被分割成两段。接着就可以拖动新节点到需要的位置，放开节点即可，如图 2.19 所示。

添加分割之后，还可以取消分割。可以按住 Shift 键，在已经分割的地方单击，分割点就会消失。分割之后，可以修改连线的形状，如图 2.20 所示。

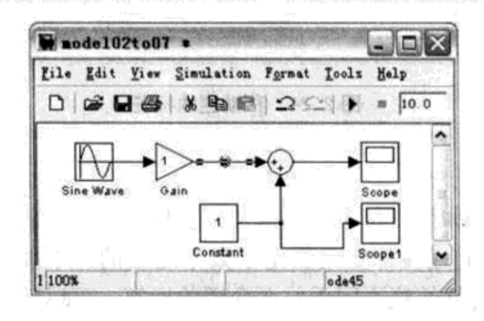


图 2.19 分割连线

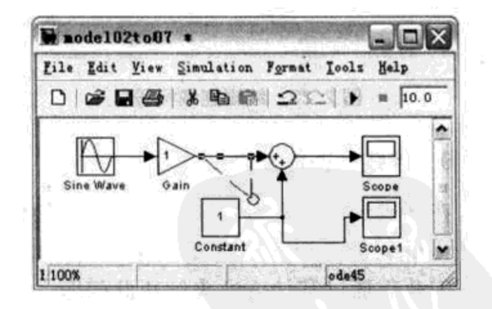


图 2.20 移动分割连线后的效果

6. 连线标签的添加

添加连线标签，有利于标明连线的功能，标签可以放在连线的任何位置。

在想要添加标签的连线上双击，连线相应的地方会出现一个编辑框。在这个编辑框中输入文本，如图 2.21 与图 2.22 所示。

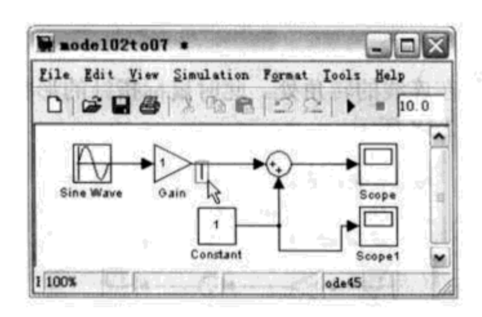


图 2.21 添加连线标签

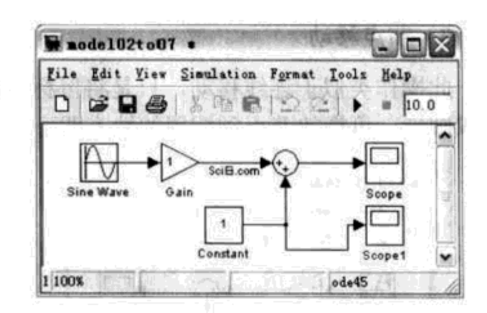


图 2.22 添加连线标签后

7. 连线标签的编辑

添加完标签后还可以编辑。将鼠标指针放在要编辑的文本上方，然后单击，就呈现可编辑状态，如图 2.23 所示，此时就可以修改已经存在的标签。

8. 连线标签的移动

除了可编辑已有标签外，还可以移动已存在的标签。将鼠标指针放在要编辑文本附近(不要放在文本的上方，但在编辑框内)，然后单击，标签就会呈被选状态，但还不能被编辑，如图 2.24 所示。比较图 2.23 和图 2.24 的标签选择状态，其中图 2.23 有光标闪动，而图 2.24 无光标闪动。

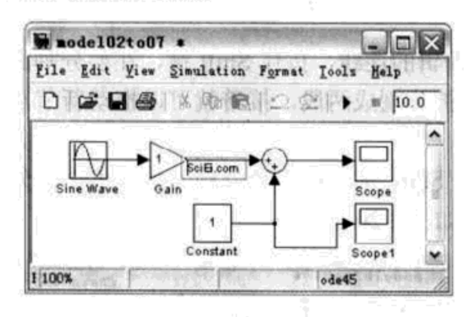


图 2.23 编辑连线标签

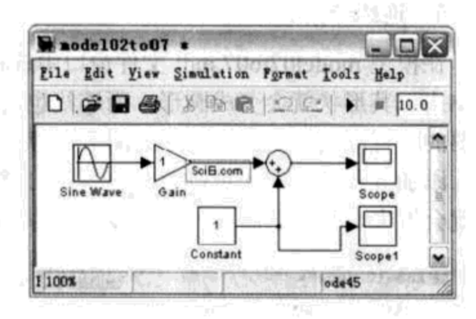


图 2.24 移动连线标签

9. 连线标签的复制

在建模过程中，有可能会遇到重复的标签，如果对每个标签都按部就班地编辑，就显得非常麻烦、费时，还容易出错。而原本存在的连线标签是固定于某个连线的，不能复制。因此需通过其他方法来解决此问题。此处的标签略不同于前面介绍的标签，可以随意被移动。

下面介绍 4 种方法来复制标签，最后得到的结果如图 2.25 所示。这 4 种方法都需要在窗口的空白区域内双击，并在出现的编辑框中输入需要创建的标签，然后在下列方法复制。

- 单击所要复制的标签，选择 Edit | Copy 命令，再选择 Edit | Paste 命令。
- 单击所要复制的标签，按 Ctrl+C 组合键，然后按 Ctrl+V 组合键。

- 单击所要复制的标签，按住 Ctrl 键，然后用鼠标拖动要复制的标签。
- 用鼠标的右键拖动要复制的标签。然后将复制的标签拖放到需要的连线上。

说明： 此方法复制的标签不能用于下面要介绍的标签传递，实质上只是一个文本。

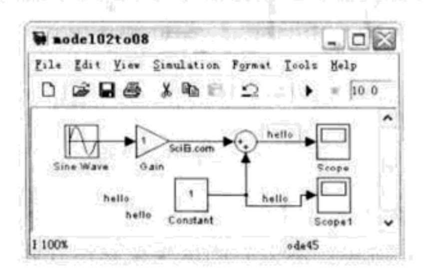


图 2.25　复制标签

10. 连线标签的传递

连线的标签可以传递，并可进行组合和分开。例如模块 Mux 与 Demux，Goto 与 From，Input 和 Output 等。通过连线标签的组合和分开，可以提高连线的可读性。

建立模型 model02to09.mdl，模块分别来自于模块集 Signal Routing，Sink 和 Sources 模块库中，并建立 Sin A 和 Sin B 输出连线标签，建立 Mux 和 Demux 模块间连线的标签，双击连线，输入小于号“<”，结果会变成图 2.26 所示。然后选择 Edit | Update Diagram 命令，结果如图 2.27 所示。在需要传递的任何连线上，只要按照上述方法，都可以得到传递标签。

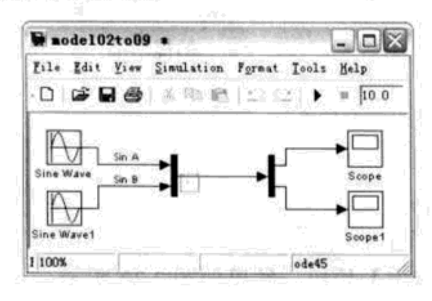


图 2.26　标签传递前模型窗口

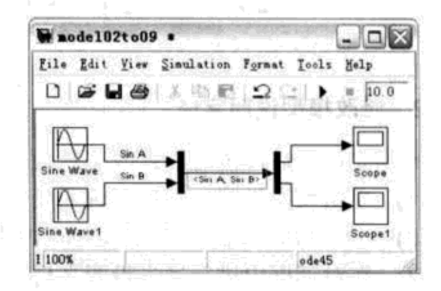


图 2.27　执行传递后的标签

2.4 模型说明

使用模型说明可以让模型更加易懂，主要目的就是说明模型的功能和使用方法。对使用 Simulink 来说，这是非常重要的。

1. 添加模型说明

在模型窗口中，可以在任何位置双击，此时就会出现一个可编辑框。新建模型 model02to10.mdl，双击模型窗口空白处，会出现一个编辑框，输入需要添加的内容，结果如图 2.28 所示。

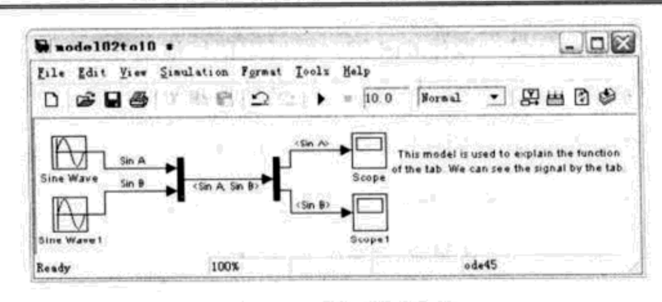


图 2.28 添加模型说明

说明： 模型说明只能为英文，不能含有汉字，暂时不支持汉字文本。如果带有汉字，如图 2.29 所示，本书的测试模型在保存此类模型时会弹出一个错误对话框，说明不能够被保存，如图 2.30 所示。

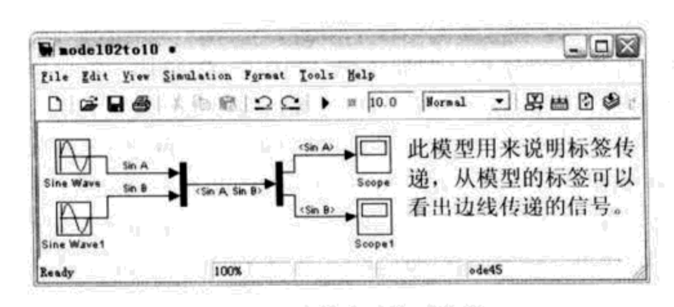


图 2.29 添加中文模型说明



图 2.30 保存中文模型说明时弹出的提示框

2. 修改模型说明字体

添加完说明后，还可以修改字体及其大小，操作步骤如下：

(1) 单击模型说明，使模型说明处于被选状态。

(2) 选择 Format | Font 命令，弹出如图 2.31 所示对话框。

(3) 在对话框中进行适当设置之后，单击【确定】按钮，结果如图 2.32 所示。

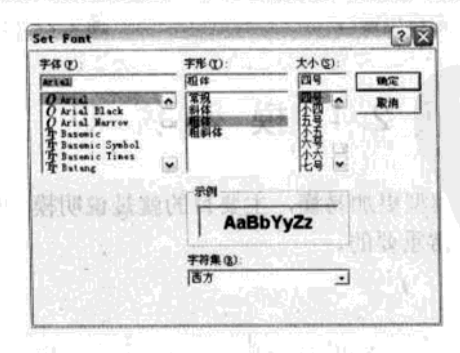


图 2.31 字体设置对话框

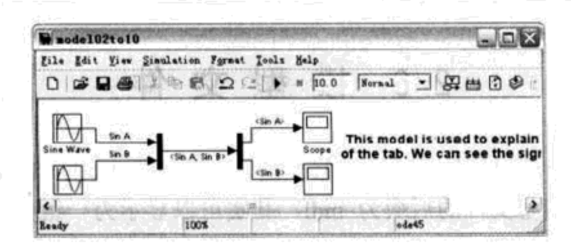


图 2.32　修改后的模型说明字体

2.5 模 型 打 印

有时候需要将模型输出到打印机上以便打印出来检查，打印方法主要有 3 种：菜单打印、粘贴到文档中和使用 MATLAB 中的 print 命令。

1. 菜单打印

使用菜单打印的操作步骤如下：

(1) 选择 File | Print Setup 命令，弹出如图 2.33 所示对话框，可设置各种打印属性，设置好以后单击【确定】按钮。

(2) 选择 File | Print 命令，弹出如图 2.34 所示，然后单击 OK 按钮。



图 2.33　模型打印设置

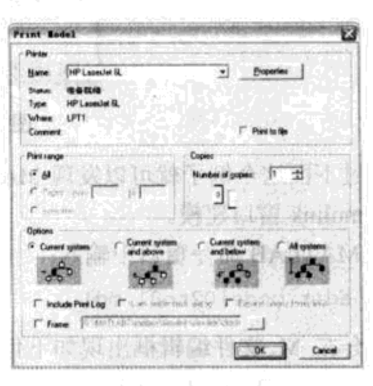


图 2.34　图形打印

2. 嵌入文档中打印

这主要是以图片形式嵌入文档(如 Word)中的打印，主要有两种方法：

- 在模型窗口中选择 Edit | Copy Model To Clipboard 命令，这样模型就被复制到剪贴板中，然后粘贴到文档中就 OK 了。
- 用抓图的方法，使用键盘中 Print Screen Sys Rq 键，然后粘贴到图形处理程序中，进行适当的处理，或用其他抓图软件处理也可以，然后打印。

3. 使用 MATLAB 的 print 命令

使用 print 命令可以将图形输出到打印机、剪贴板或其他文档中。print 命令的具体语法和其他功能可用 help print 查看，在此不再赘述。

2.6 模型文件

Simulink 中的各个模型都是可以适当修改的，不过这需要用户对 Simulink 有非常深入的了解。Simulink 的模型文件的后缀为 mdl，其实可以显示为一组代码，主要有以下几个部分组成：

- Model section：定义模型的参数。
- BlockDefaults sections：模块的默认设置。
- AnnotationDefaults sections：模型注解的默认值。
- System sections：顶层系统或者子系统的描述参数，包括 line，block 和 annotation 等。

下面通过一个简单实例来看看具体代码文件的内容。实例如图 2.35 所示，保存文件名为 model02to01.mdl。

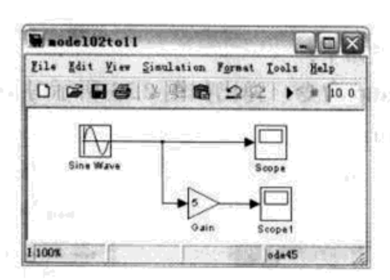


图 2.35 简单的模型

通过下面这个例子就可以发现 MATLAB 和 Simulink 这两种模型形式的不同，最后会选用 Simulink 窗口建模。

在 MATLAB 命令窗口中输入：

```
>> edit model02to01.mdl
```

则会在 M 文件编辑框出现如下代码，代码的具体含义可以参照模型来理解。这些代码就是以 M 文件形式打开的 model02to01.mdl 文件。由于程序过长，在此不详细列出，源程序共约 500 行，共 13 页。

model02to01.mdl 程序代码如下：

```
Model {
  Name              "model02to01"
  Version           6.0
  GraphicalInterface {
    NumRootInports        0
    NumRootOutports       0
    ParameterArgumentNames  ""
    ComputedModelVersion    "1.4"
    NumModelReferences      0
    NumTestPointedSignals   0
  }
  SavedCharacterEncoding  "ibm-1386_P100-2002"
  SaveDefaultBlockParams  on
```

```
SampleTimeColors    off
LibraryLinkDisplay      "none"
WideLines       off
ShowLineDimensions      off
ShowPortDataTypes   off
ShowLoopsOnError    on
IgnoreBidirectionalLines off
ShowStorageClass    off
ShowTestPointIcons      on
ShowViewerIcons     on
SortedOrder         off
ExecutionContextIcon    off
ShowLinearizationAnnotations on
RecordCoverage      off
CovPath         "/"
CovSaveName         "covdata"
CovMetricSettings   "dw"
CovNameIncrementing     off
CovHtmlReporting    on
covSaveCumulativeToWorkspaceVar on
CovSaveSingleToWorkspaceVar on
CovCumulativeVarName    "covCumulativeData"
CovCumulativeReport     off
CovReportOnPause    on
ScopeRefreshTime    0.035000
OverrideScopeRefreshTime on
DisableAllScopes    off
DataTypeOverride    "UseLocalSettings"
MinMaxOverflowLogging   "UseLocalSettings"
MinMaxOverflowArchiveMode "Overwrite"
BlockNameDataTip    off
BlockParametersDataTip  off
BlockDescriptionStringDataTip off
ToolBar         on
StatusBar       on
BrowserShowLibraryLinks off
BrowserLookUnderMasks   off
Created         "Fri Feb 18 22:13:03 2005"
UpdateHistory       "UpdateHistoryNever"
ModifiedByFormat    "%<Auto>"
LastModifiedBy      "ChinaMaker"
ModifiedDateFormat      "%<Auto>"
LastModifiedDate    "Fri Feb 18 23:19:36 2005"
ModelVersionFormat      "1.%<AutoIncrement:4>"
ConfigurationManager    "None"
LinearizationMsg    "none"
Profile         off
ParamWorkspaceSource    "MATLABWorkspace"
AccelSystemTargetFile   "accel.tlc"
AccelTemplateMakefile   "accel_default_tmf"
AccelMakeCommand    "make_rtw"
TryForcingSFcnDF    off
ExtModeBatchMode    off
ExtModeEnableFloating   on
ExtModeTrigType     "manual"
ExtModeTrigMode     "normal"
ExtModeTrigPort     "1"
ExtModeTrigElement      "any"
ExtModeTrigDuration     1000
ExtModeTrigDurationFloating "auto"
ExtModeTrigHoldOff      0
```

```
ExtModeTrigDelay    0
ExtModeTrigDirection    "rising"
ExtModeTrigLevel    0
ExtModeArchiveMode      "off"
ExtModeAutoIncOneShot   off
ExtModeIncDirWhenArm    off
ExtModeAddSuffixToVar   off
ExtModeWriteAllDataToWs off
ExtModeArmWhenConnect   on
ExtModeSkipDownloadWhenConnect off
ExtModeLogAll       on
ExtModeAutoUpdateStatusClock on
BufferReuse         on
ProdHWDeviceType    "32-bit Generic"
ShowModelReferenceBlockVersion off
ShowModelReferenceBlockIO off
Array {
  Type        "Handle"
  Dimension           1
  Simulink.ConfigSet {
    $ObjectID           1
    Version             "1.0.4"
```
......省略部分......
```
LineDefaults {
  FontName        "Helvetica"
  FontSize        9
  FontWeight          "normal"
  FontAngle           "normal"
}
System {
  Name        "model02to01"
  Location        [203, 353, 511, 504]
  Open        on
  ModelBrowserVisibility  off
  ModelBrowserWidth       200
  ScreenColor         "white"
  PaperOrientation    "landscape"
  PaperPositionMode       "auto"
  PaperType           "A4"
  PaperUnits          "centimeters"
  ZoomFactor          "100"
  ReportName          "simulink-default.rpt"
  Block {
    BlockType           Constant
    Name            "Constant"
    Position            [64, 15, 106, 125]
    Orientation         "up"
    NamePlacement       "alternate"
    Value           "9999.9999"
  }
 }
}
```

第 3 章　Simulink 运行仿真

运行 Simulink 模型的仿真模式主要有两种：Simulink 模型窗口运行模式和 MATLAB 命令窗口仿真模式。这两种方法的区别类似于 Windows 操作系统和 Dos 操作系统的区别，前者直接在 Simulink 窗口通过鼠标进行操作，后者在 MATLAB 命令窗口通过命令的形式对模型进行仿真。前者比后者直观，后者比前者容易进行批处理。

本章主要讲解 Simulink 窗口仿真模式，MATLAB 命令窗口仿真模式将在第 10 章进行详细讲解。

本章主要内容包括：

- Simulink 模型窗口运行模式
- 设置仿真性能与计算精度

3.1　Simulink 模型窗口运行模式

3.1.1　窗口仿真基本操作

直接使用 Simulink 窗口方式进行仿真交互性，操作简单明了，不需了解这些操作所执行的具体命令及其语法。例如最简单、最常用的 ODE45 命令：

```
[T,Y] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS,P1,P2...)
```

可以看出，要想记住命令中的参数是非常困难的，不利于初学者使用，而且在仿真过程中也不能修改参数。MATLAB 命令模式形式也并非一无是处，事实上，命令窗口仿真也有它的优点，即可以同时处理一批系统模型的仿真。

上面提到过，在仿真过程中可修改模型参数，但是下面情况除外：

- 采样时间
- 模型经过零点个数
- 模块中的参数维数
- 模型的状态
- 模型的输入输出个数
- 内部模块工作向量的维数
- 增加和删除模块
- 增加和删除信号线

如果要完成这些修改，必须停止模型仿真，修改完成后再进行仿真。

利用 Simulink 窗口进行仿真，主要有以下几个操作：

- 设置仿真参数
- 运行仿真
- 终止仿真
- 暂停仿真

● 仿真诊断

下面详细介绍这几个主要操作：

1. 设置仿真参数

在 Simulink 模型窗口中选择 Simulation | Configuration Parameters 命令，弹出如图 3.1 所示的仿真参数设置对话框。图中左侧列表框中的目录树包括 Solver、Data Import/Export、Optimization、Diagnostics、Hard ware Implementation、Model Referencing 和 Real-Time Workshop 等几项。右侧是每一项所包含的参数设置选项。

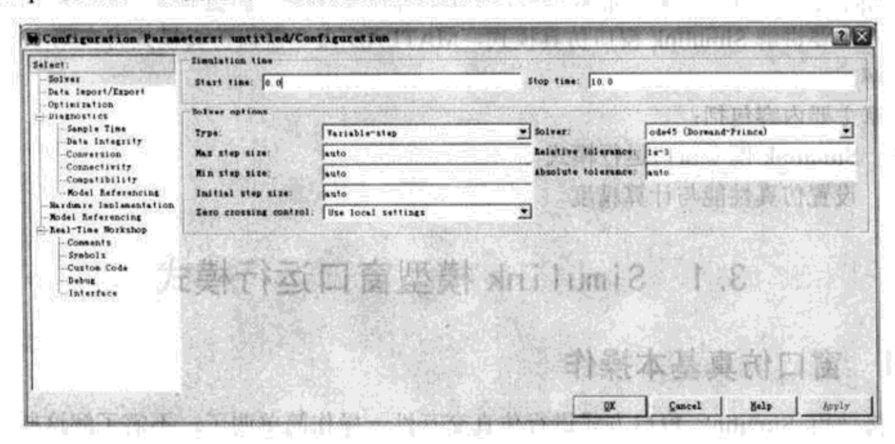


图 3.1 仿真参数设置对话框

仿真参数设置好之后，单击 Apply 按钮应用设置，或者单击 OK 按钮。前者应用设置，但是不关闭对话框，后者应用并关闭对话框。

说明： 单击 Cancel 按钮表示不设置参数并关闭对话框，单击 Help 按钮显示对话框的帮助文档。

2. 运行仿真

设置好 Simulink 模型运行环境之后，可以运行仿真了。选择 Simulation | Start 命令运行仿真，或者使用 Ctrl+T 快捷键，或者单击▶按钮直接运行。模型运行时，命令 Simulation | Start 自动变化为 Simulation | Stop，运行按钮▶变为暂停按钮⏸。

3. 终止仿真

与运行仿真操作类似，当运行仿真后可选择 Simulation | Stop 命令，或者使用快捷键 Ctrl+T，或者单击按钮■直接终止。

4. 暂停仿真

与终止仿真操作类似，当运行仿真后可选择 Simulation | Pause 命令，或者单击⏸按钮直接暂停。

5. 仿真诊断

在仿真过程中，如果模型中存在错误，运行会被终止，并弹出 cmacpid(诊断)对话

框，在对话框中显示错误信息，如图 3.2 所示。错误信息分为上下两部分，上部分为出错模块的详细信息。错误信息及其所表示的含义如表 3.1 所示。

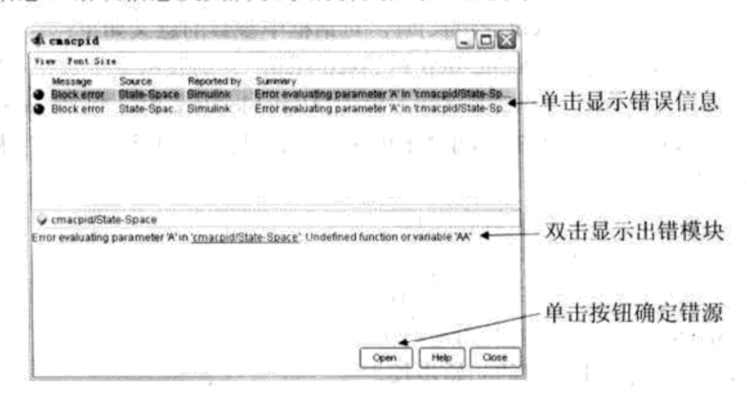


图 3.2　仿真诊断对话框

表 3.1　错误信息介绍

错误信息	信息含义
Message	信息类型，如错误模块，警告，日志
Source	导致出错的元素名，如模块
Reported by	信息来源，如 Simulink, Stateflow, Real-Time Workshop
Summary	出错信息摘要

3.1.2　仿真参数设置

Configuration Parameters 对话框中各个参数包括 Solver、Data Import/Export、Optimization、Diagnostics、Hardware Implementation、Model Referencing 和 Real-Time Workshop 等。

1. Solver 求解器

求解器设置如图 3.3 所示，包括两个选项组 Simulation time 和 Solver options，可以设置仿真的起止时间、求解器类型、误差大小等。

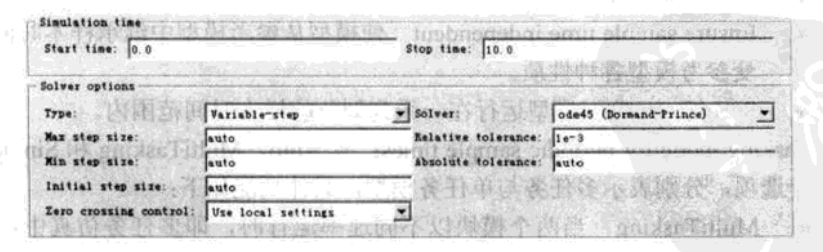


图 3.3　Solver 求解器参数

1) Simulation time 仿真起止时间设置

- Start time：仿真起始时间，默认为 0。

- Stop time：仿真终止时间，默认为 10。

2) Solver options 仿真求解器设置

当 Type 选项为 Variable-step(变步长)时，对话框中参数的意义如下：

- Type：此选项包括 Variable-step 和 Fixed-step，分别表示变步长和定步长。
- Solver：表示求解方法，当 Type 值为 Variable-step 时，包括 ode45、ode23、ode113、ode15s、ode23s、ode23t 和 ode23tb，其中前 3 个为非刚性求解方法，其余为刚性求解方法。
- Max step size：求解时的最大步长。
- Min step size：求解时的最小步长。
- Relative tolerance：求解时的相对误差。
- Absolute tolerance：求解时的绝对误差。
- Initial step size：求解时的初始步长。
- Zero crossing control：在变步长仿真中打开零交叉检测功能。对于大多数模型，此功能可以增加时间步长从而加速仿真。如果模型动态变化剧烈，关闭这个选项能够加速仿真，但是降低了仿真的精度。

当 Type 选项为 Fixed-step(定步长)时，如图 3.4 所示对话框中参数的意义如下。

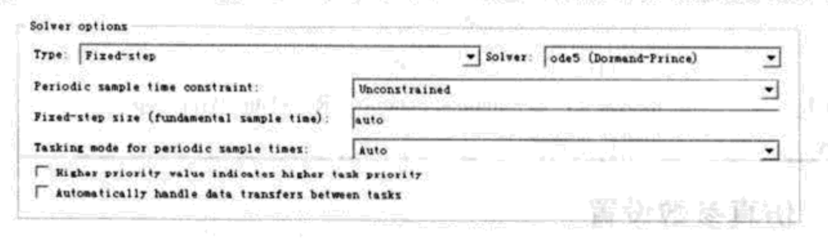


图 3.4 固定步长时的参数

- Solver：当 Type 值为 Fixed-step 时，求解方法包括 ode1、ode2、ode3、ode4、ode5、ode14x。
- Periodic sample time constraint：允许指定模型样本周期限制，在模型仿真过程中，Simulink 会确保满足此要求，如果不满足要求，会出现错误信息。此参数包括以下选项：
  - ◆ Unconstrained　无限制；
  - ◆ Ensure sample time independent　使模型从参考模型中继承样本时间，而不改变参考模型各种性质。
  - ◆ Specified　确保模型运行在一系列划分的样本时间范围内。
- Tasking mode for periodic sample times：有 Auto，MultiTasking 和 SingleTasking 3 个选项，分别表示多任务与单任务模型，具体功能如下：
  - ◆ MultiTasking　当两个模块以不同速率运行时，即多任务仿真中，这种模式将会在检测到两个模块间出现不合法的样本速率时发出错误信号。不合法的样本速率会导致一个任务的输出数据不能被另外一个任务使用。通过检测这种信号传输，MultiTasking 模式可以帮助创建一个合法的现实多任务系统。

- SingleTasking 这种模式不会检测模块间的信号传输，当模型是一个单任务模型时，所有的信号传输都是同步的，不需要检测。
- Auto 这种模式会自动选择不同的运行模式，当模型是单任务模型时就用SingleTasking模式，当是多任务模型时就会自动选择MultiTasking模式。

其下还包括两个复选框，功能如下：

- Higher priority value indicates higher task priority 如果选中该复选框，模型的目标实时系统将会给不同的任务分配不同的优先权，高的优先权分配给高优先值的模块。也就是说会导致模型中低优先值的模块与高优先值的模块间的信号传输是异步的。如果不选中该复选框，目标实时系统将为低优先值任务分配更高的优先值。详细内容可参看Real-Time Workshop帮助文档。
- Automatically handle data transfers between tasks 如果选中该复选框，将会在模块间插入隐含速率传输模块。

2. Data Export/Import——数据输出输入

单击Configuration Parameters对话框左侧目录中的Data Export/Import选项，右侧如图3.5所示。

Load from workspace
Input: [t, u]
Initial state: xInitial
Save to workspace
Time: tout
States: xout
Output: yout
Final states: xFinal
Save options
Limit data points to last: 1000 Decimation: 1
Format: Array Signal logging name: logsout

图3.5 数据输出输入参数

1) Load from workspace：包含若干控制选项，可以设置如何从MATLAB工作空间调入数据。

- Input：格式为MATLAB表达式，确定从MATLAB工作空间输入的数据。
- Initial state：格式为MATLAB表达式，确定模型的初始状态。

2) Save to workspace：可以设置如何将数据保存到MATLAB工作空间。

- Time：设置将模型仿真中的时间导出到工作空间时所使用的变量名。
- States：设置将模型仿真中的状态导出到工作空间时所使用的变量名。
- Output：设置将模型仿真中的输出导出到工作空间时所使用的变量名。
- Final states：设置将模型仿真结束时的状态导出到工作空间时所使用的变量名。

3) Save options：包含若干控制选项，允许设置保存到工作空间或者从工作空间加载数据的各种选项。

- Limit data points to last：限制导出到工作空间的数据个数，如N。在仿真结束时，MATLAB工作空间只包含最后N个数据。

- Decimation：如果指定为 M，Simulink 则会每隔 M 个数据输出一个。
- Format：设置保存到工作空间或者从工作空间载入数据的格式，包括矩阵，结构体，带有时间的结构体。
- Signal logging name：用来保存仿真过程中信号记录的变量名。

3. Optimization 优化选项

单击 Configuration parameters 对话框左侧目录中的 Optimization 项，右侧如图 3.6 所示。这个优化选项组可以选择不同的选项来提高仿真性能以及产生代码的性能。

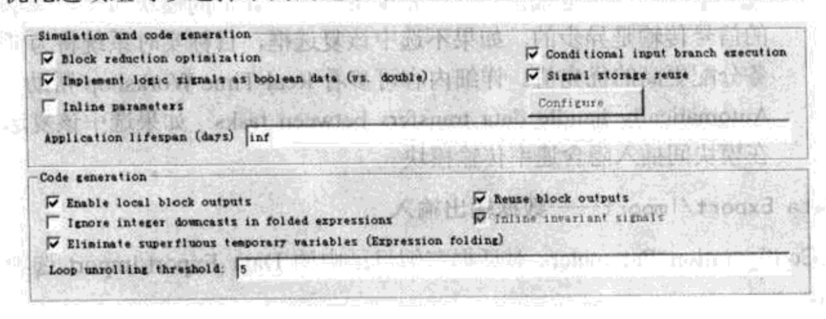


图 3.6 Optimization 参数

1) Simulation and code generation：该选项组的设置对模型仿真和代码生成共同有效。

- Block reduction optimization：用一个合成模块来代替一组模块，以此来提高模型的执行效率。
- Conditional input branch execution：该选项在模型中含有 Switch 模块或者 Multiport 模块时使用。当被选中时，该选项只执行模型中那些需要计算控制输入的，以及每一个时间步长内控制输入所选择的输入数据的 Switch 或者 Multiport Switch 模块。在通过 Real-Time Workshop 生成模型代码时具有类似的功能，以提高执行速度。
- Signal storage reuse：促使 Simulink 重新使用分配的内存来保存模块的输入与输出数据。如果不选中该复选框，Simulink 将会为每一个模块输出分配一个独立的内存，这在模型很大时将会极大地占用内存空间，因此在对模型进行调试时应该选中此复选框。如果要进行 C-MEX 函数调试，或者使用了 Floating Scope 和 Display 模块时，则不选中该复选框。
- Inline parameters：默认在仿真过程中可以修改许多可调模块参数。在仿真过程中参数可以修改的模块称为可调模块。选中此复选框，使所有模块都称为不可调模块，可以移动这些模块到仿真循环外部，从而加快模型仿真的速度以及模型代码的运行速度。
- Application lifespan (days)：设置模型所代表系统的活动周期。这个参数和仿真步长决定了用来保存绝对时间值的固定点模块的数据类型。

2) Code generation：该选项组的设置只对代码生成有效。

4. Diagnostics 诊断

Diagnostics 参数配置控制面板可以配置适当的参数，如图 3.7 所示，以便在仿真执行过程中遇到异常条件时采取相应的措施。

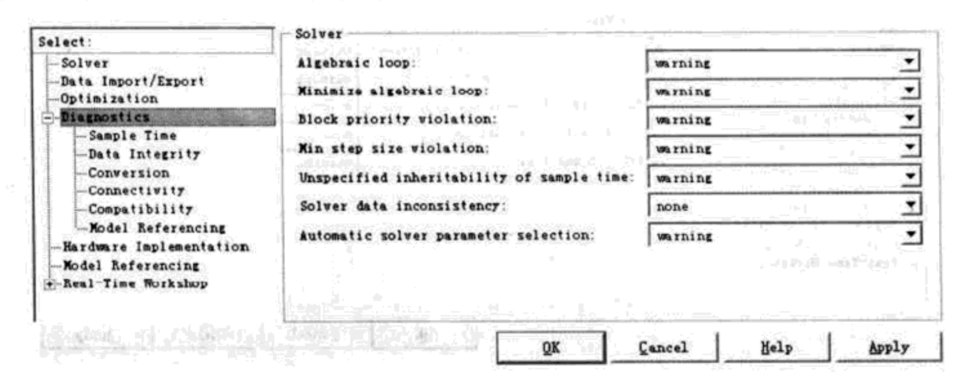


图 3.7　Diagnostics 参数

1) Solver：当 Simulink 检测到与求解器相关的错误时，这个控制组可设置诊断措施。

- Algebraic loop：在执行模型仿真时可以检测到代数环。共有 3 个参数可供选择，none、warning 和 error，如果选择 error，Simulink 将会显示错误信息并高亮显示组成代数环的模块。选择 none 则不给出任何信息及提示，选择 warning 会给出相应的警告而不会中断模型的仿真。
- Minimize algebraic loop：如果需要 Simulink 消除包含有子系统的代数环及这个子系统的直通输入端口，就可以设置此选项来采取相应的诊断措施。如果代数环中存在一个直通输入端口，仅当代数环所用的其他输入端口没有直通时，Simulink 才可以消除这个代数环。
- Block priority violation：当仿真运行时，Simulink 检测优先设置错误选项的模块。
- Min step size violation：允许下一个仿真步长小于模型设置的最小时间步长。当设置的模型误差需要的步长小于设置的最小步长时，此选项起作用。
- Unspecified inheritability of sample time：当模型中包含有 S 函数，但又不排除函数从父模型中继承样本时间时，指定诊断时采取的应对措施。仅当仿真过程中使用的是固定步长的离散求解器，以及求解器有周期样本时间限制时 Simulink 才会检测。
- Solver data inconsistency：兼容性检测是一个调试工具，确保满足 Simulink 中 ODE 求解器的若干假设。其主要作用是让 S 函数和 Simulink 的内部模块具有同样的执行规则。由于兼容性检测会导致仿真性能的大大降低，甚至可达到 40%，一般这个选项都设置为 none。利用兼容性检测来检测 S 函数，有助于找到出现非预期仿真结果的原因。
- Automatic solver parameter selection：当 Simulink 改变求解器参数时采取的诊断措施。例如，假如用一个连续求解器来仿真离散模型，并设置此选项为 warning，此时，Simulink 就会改变求解器的类型为离散，并在 MATLAB 命令窗口显示一

个有关于此的警告信息。

2) Sample Time：当 Simulink 检测到与模型样本时间相关的编辑错误时，这个控制组可以设置诊断措施，如图 3.8 所示。

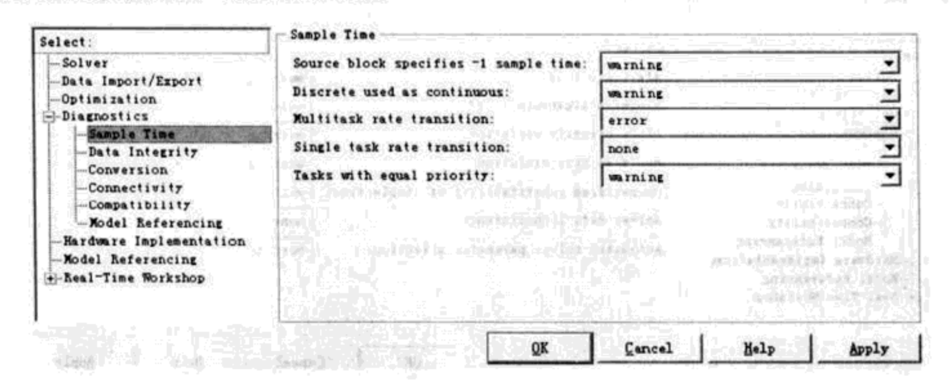


图 3.8 设置 Sample Time 参数

- Source block specifies -1 sample time：设置源模块的样本时间为-1，如 Sine Wave 模块。
- Discrete used as continuous：将离散模块作为连续模块，例如，单位延迟模块是一个离散模块，但是可从与其输入端相连的模块处继承连续样本属性。
- Multitask rate transition：在多任务模式中的两个模块，会出现两个无效速率的转换。
- Single task rate transition：在单任务模式中，两个模块间的速率会进行转换。
- Tasks with equal priority：这个模型所表示的目标中的一个异步任务与另外一个目标异步任务具有同样的优先级。如果目标允许具有同样优先级的任务相互支配，则必须将选项设置为 error。

3) Data Integrity：数据完整性诊断。设置当 Simulink 检测到有危害模型定义的数据完整性条件时，Simulink 所采取的诊断措施，如图 3.9 所示。

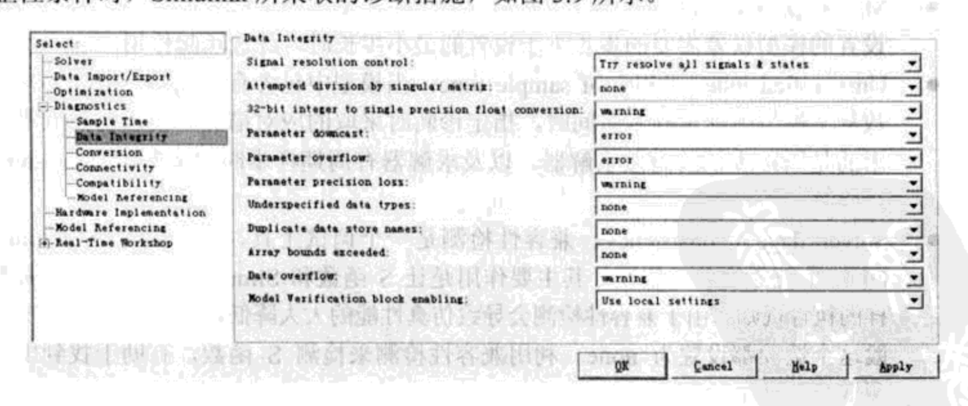


图 3.9 设置 Data Integrity 参数

- Signal resolution control：设置 Simulink 如何求解传向 MATLAB 工作空间 Simulink.Signal 对象的信号。其中包括以下选项。

◆ Try resolve all signals & states (warn for implicit resolution)：将求解每一个传向 Simulink.Signal 目标的信号或者状态，这些信号或者状态具有与 Simulink.Signal 同样的名称。如果信号或状态隐性地求解到信号对象就会显示警告信息，例如，一个信号对象具有与 MATLAB 工作空间中已经存在的信号或状态同名，而模型又没有设置这些信号或者状态应该传输到信号对象。
◆ Try resolve all signals & states：传输每一个信号或者离散状态到具有同名的 Simulink.Signal 对象中，而不管模型是否已经设置这些信号或者状态到信号对象中。
◆ Use local settings：求解每一个模型指定的信号或者离散状态到 MATLAB 工作空间 Simulink.Signal 中。

- Attempted division by singular matrix：Product 模块通过对相乘的输入矩阵进行求逆来检测是否存在奇异矩阵。
- 32-bit integer to single precision float conversion：32 位整数转换为浮点值，这个转换会带来精度的损失。
- Parameter downcast：将模块输出的参数类型转化到具有更小值域的参数类型，例如将 uint32 转换到 uint8，这个诊断仅仅应用于可调谐的参数。
- Parameter overflow：参数溢出，参数的数据类型不能容纳参数值。
- Parameter precision loss：将模块输出转换到低精度的数据类型，例如，将 double 转换为 uint8。
- Underspecified data types：不设置数据类型。Simulink 在数据传播过程中不能判断数据的类型。
- Duplicate data store names：复制数据保存所使用的变量名。
- Array bounds exceeded：这个选项将会促使 Simulink 在仿真过程中检测模块是否写到所分配内存的外部。最典型的就是当用户自己编写的 S 函数存在一个漏洞时就会发生这种情况。如果是激活状态，将在模块每次执行时检测每一个模块，直接产生的影响就是降低了模型执行的速度。为了避免不必要的降低执行速度，只有当怀疑模型中编写的 S 函数存在问题时激活这个功能。
- Data overflow：表示信号或参数的值超过信号或参数数据类型所能够保存的值。
- Model Verification block enabling：这个参数可以全局，或者局部激活，或者非激活模型中的模型验证(verification)模块。其中可以选择如下选项：
  ◆ Use local settings　根据每个认证模块的激活判定参数值来激活或者非激活模块。如果模块的激活判定值为 on，那么模块就处于激活状态，否则相反。
  ◆ Enable all　不管模块的激活判定是如何设置，将所有的认证模块都激活。
  ◆ Disable all　不管模块的激活判定是如何设置，将所有的认证模块都非激活。

4) Conversion：该选项组用于用户设置诊断，以便在模型编译过程中 Simulink 检测到模型中存在数据转换问题时所采取的应对措施，如图 3.10 所示。

- Unnecessary type conversions：将 Data Type Conversion 模块添加到不需要转换数据的地方。
- Vector/matrix block input conversion：在模块输入端口会进行向量到矩阵或者矩阵

到向量的转换。

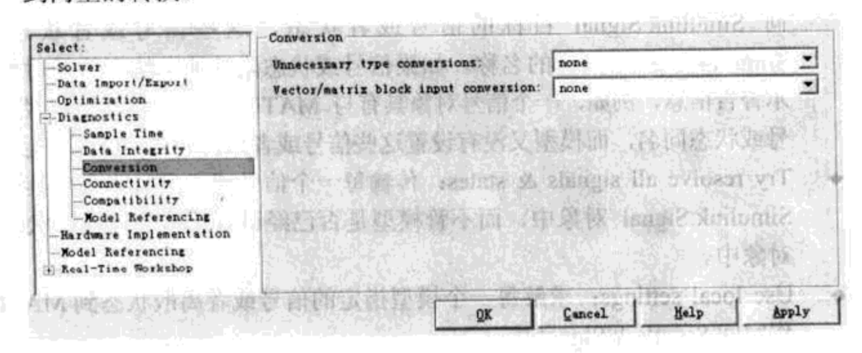


图 3.10　设置 Conversion 参数

5) Connectivity：这个选项组可以设置相应的诊断，以便在模型编译过程中 Simulink 检测到模块的连接问题时采取相应的措施，如图 3.11 所示。

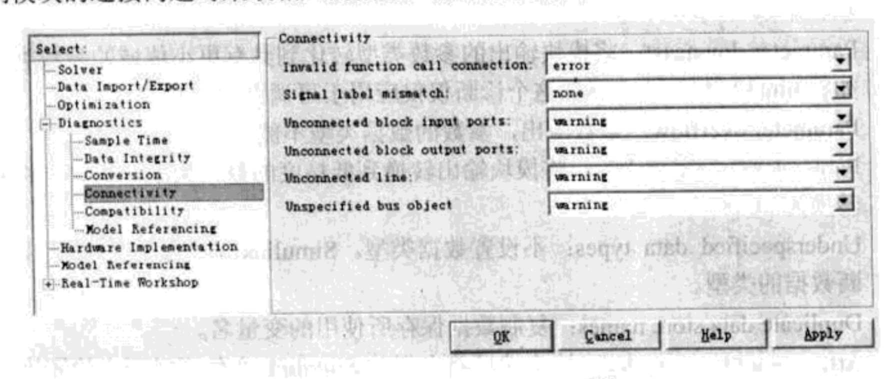


图 3.11　设置 Connectivity 参数

- Invalid function call connection：Simulink 检测到模型中存在一个不正确的函数回调子系统。如果不激活这个错误信息可能会得到一个错误的仿真结果。
- Signal label mismatch：仿真中会遇到虚拟信号，它们具有同一个信号，但是具有不同的标签。
- Unconnected block input ports：模型中包含一个模块，其有一个输入端口没有信号线与之连接。
- Unconnected block output ports：模型中包含一个模块，其中有一个输出端口没有信号线与之连接。
- Unconnected line：模型中含有一个没有连接的信号线。
- Unspecified bus object：设置相应的诊断，当 Simulink 遇到此类问题时所采取的措施，即当为参考模型生成仿真目标时，如果模型中的任何一个 Outport 模块都连接到总线，但是没有设置总线对象。

6) Compatibility：该选项组允许用户设置相应的诊断措施，以便在模型升级或者模型仿真过程中，检测到 Simulink 不同版本之间的不兼容性时采取相应的应对措施，如图 3.12 所示。

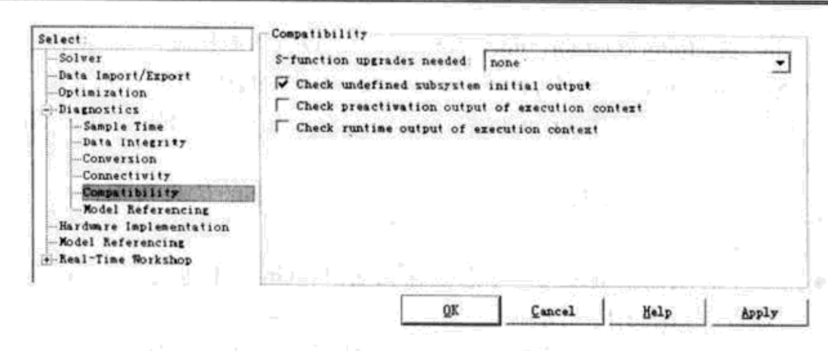


图 3.12　设置 Compatibility 参数

7)　Model Referencing：该选项组允许用户设置相应的诊断措施，以便在模型升级或者模型仿真过程中，检测到 Simulink 不同版本之间的不兼容性时采取相应的应对措施。功能类似于 Compatibility Diagnostics，只是针对的对象有所不同，如图 3.13 所示。

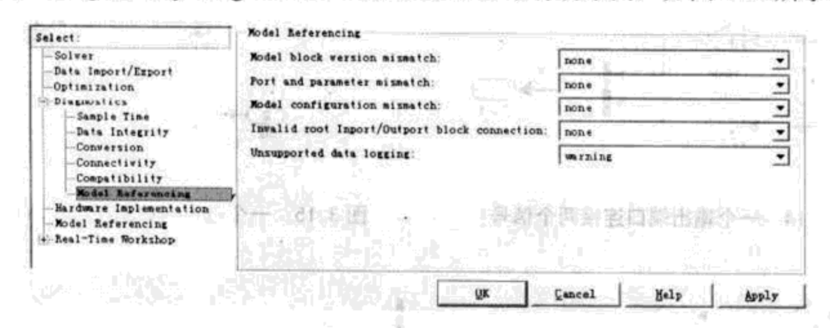


图 3.13　设置 Model Referencing 参数

- Model block version mismatch：设置诊断措施，在模型加载或更新升级过程中，当 Simulink 检测到用来创建或更新 Model 模块的两个模型版本不兼容时所采取的对应措施。其中的选项包括：
  - ◆ none　默认。
  - ◆ warning　刷新 Model 模块,同时报告警告信息。
  - ◆ error　显示错误信息，但是不刷新 Model 模块。
- Port and parameter mismatch：设置诊断措施，在模型加载或者更新升级过程中，当 Simulink 检测到本模型中 Model 模块的 I/O 端口和参考模型的总线 I/O 端口之间的失配，或者是本模型中 Model 模块的参数与参考模型中的参数失配时采取相应的应对措施。其中的选项包括：
  - ◆ none　默认
  - ◆ warning　刷新过时的 Model 模块,并显示警告信息。
  - ◆ error　显示错误信息，但是不刷新过时的 Model 模块。
- Model configuration mismatch：设置诊断措施，在当前模型与参考模型的参数设置不匹配，或者参考模型本身的参数设置存在问题时所应当采取的应对措施。默认设置为 none。如果怀疑模型中存在失配的参数设置并可能导致错误的结果时，可以设置诊断为 warning 或者 error。

- Invalid root Inport/Outport block connection：设置诊断措施，在代码生成过程中，当 Simulink 检测到有不合法的内部连接到模型的总线输出模块端口时所采取的应对措施。当设置为 error 时，如果遇到以下几种连接情况，Simulink 会报告错误：
    - 一个总线输出端口直接或间接的与一个以上的信号数据连接，如图 3.14 所示。
    - 一个总线输出端口与一个总线输入端口、Ground(地线)模块或者非数据模块连接，如图 3.15 所示。
    - 两个总线输出模块被连接到同一个模块端口，如图 3.16 所示。
    - 一个总线输出模块不能连接到模块某些输出部分，如图 3.17 所示。
    - 一个输出模块不能对同一个信号数据输出两次，如图 3.18 所示。

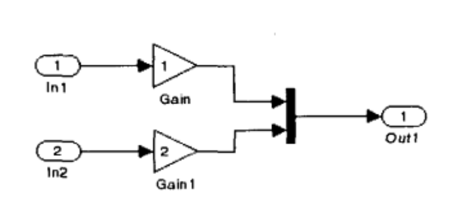


图 3.14 一个输出端口连接两个信号

图 3.15 一个输出端口连接 Ground 等模块

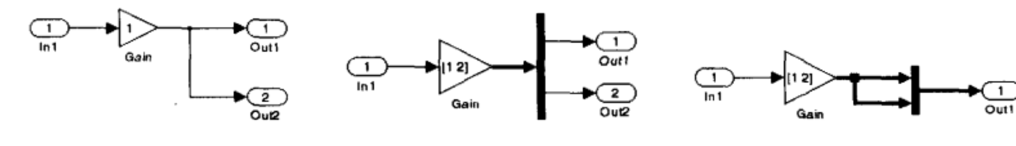


图 3.16 两总线输出连接同一模块

图 3.17 不能连接某些端口

图 3.18 不能输出两次

- Unsupported data logging：设置诊断措施，在当前模型中包含有 To Workspace 模块或者激活输出功能的 Scope 模块时所采取的应对措施。激活输出功能的 Scope 模块指设置 Scope 输出功能。默认措施为警告用户 Simulink 不支持用此类模块从参考模型中获取数据功能。

5. Hardware Implementation 硬件设置

该参数配置控制面板主要针对基于计算机系统的模型，如嵌入式控制器，如图 3.19 所示。允许设置这些用来执行模型所表示系统的硬件的参数。可以使得模型仿真中可以检测到目标硬件中存在的错误条件，如硬件的溢出。

1) Embedded hardware：该选项组可以设置硬件的特性，以便用来执行模型系统仿真。该选项组中包括以下选项。

- Device type：确定用来执行仿真硬件的类型，下拉列表中是 Simulink 所能检测到的硬件信息，因此不需要手动输入。

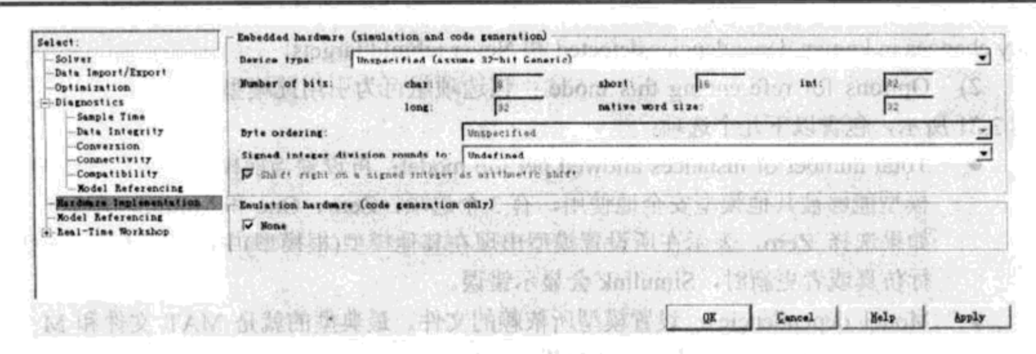


图 3.19　设置 Hardware Implementation 参数

- Number of bits：该选项组是确定所选择硬件设备所支持的 C 语言数据二进制长度。如果 Simulink 知道所选择设备所支持的数据长度，则这组选项呈现灰色不能被人为设置。
- Native word size：确定硬件设备支持的二进制命令长度。如果 Simulink 知道所选择设备所支持的命令长度，则该选项呈现灰色不能被人为设置。
- Signed integer division rounds to：确定 ANSI C 编译器是如何适应环境为硬件编译代码，使一个符号整数除以另外一个符号整数时得到一个带符号分数进行近似。
- Shift right on a signed integer as arithmetic shift：如果 C 编译器要进行一个符号整数右移运算时选中该复选框，它只影响代码生成。
- Byte ordering：设置目标硬件数据第一个字节的重要性。如果最重要，则选择 Big Endian。如果最不重要则选择 Little Endian。如果重要性未知，则默认不选择。此设置只影响代码生成。更多内容可参见 Real-Time Workshop 帮助文档。

2)　Emulation hardware：该选项组允许用户设置硬件特性来测试模型所生成的代码。

None：如果选择，则测试模型生成代码的硬件特性与原生成代码的特性一样或者设置一样。如果要用不同的硬件特性进行测试，则非选此选项，这将会扩展控制组，以便设置不同的特性，如图 3.20 所示。

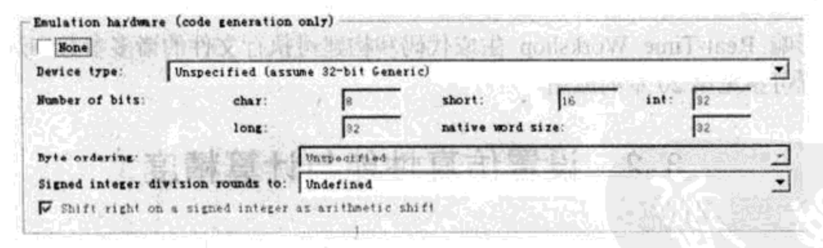


图 3.20　Emulation hardware 选项组中非选 None 的扩展面板

6. Model Referencing 参考模型

模型参考面板允许用户设置模型中的其他模型，或者包含在其他模型中的此模型，以便仿真的调试和目标代码的生成。

1)　Rebuild options for all referenced models：该选项组允许用户为直接或者间接引用的模型进行结构设置。包括 Always rebuild targets、If any changes detected (默认设置)、If

any changes in known dependencies detected 和 Never rebuild targets。

2) Options for referencing this model：该选项组可为引用此模型进行选项设置，如图 3.21 所示，包含以下几个选项：

- Total number of instances allowed per top model：可设置当前模型中有多少个参考模型能够被其他模型安全地使用，有 3 个选项：Zero、One 和 Multiple (默认)。如果选择 Zero，表示在所设置模型出现在其他模型(根模型)中时，当对根模型进行仿真或者更新时，Simulink 会显示错误。
- Model dependencies：设置模型所依赖的文件。最典型的就是 MAT 文件和 M 文件，常被用来对参数进行初始化提供数据。
- Pass scalar root inputs by value：选中该复选框，可以使其他模型通过此模型中以数值形式表示的标量调用此模型。否则，其他模型将会以引用形式调用此模型，以引用形式通过模型输入，例如，通过输入的地址而不是输入的值。
- Minimize algebraic loop occurrences：选中该复选框，可以使 Simulink 尽量减少其他模型引用此模型中的代数环。

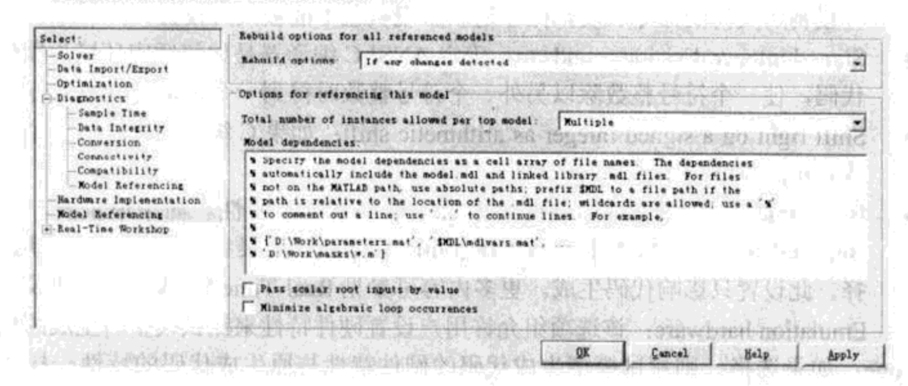


图 3.21 Options for referencing 参数对话框

7. Real-Time Workshop

设置影响 Real-Time Workshop 生成代码和构建可执行文件的诸多参数和选项。其具体参数设置可参见第 20 章的应用。

3.2 设置仿真性能与计算精度

系统的仿真性能和计算精度受到非常多因素的影响，如参数的选择、模型本身的搭建结构。通过 3.1 节的介绍，可知参数设置对话框中的若干设置可以影响到 Simulink 仿真速度，如求解器的选择、时间步长的设置、是否检测模型中的匹配问题等。

速度与精度往往呈现矛与盾的关系，此消彼涨，需要时具体的仿真对象和环境以及操作者的要求来决定。

3.2.1 Simulink 加速仿真

影响 Simulink 仿真速度的原因比较复杂，可能是由某一个原因造成的，也可能是多个原因共同而成的，需要用户不断地调试。简单列举若干可能影响仿真速度的因素如下：

- Simulink 模型可能为一个刚性方程，对于默认的 ode45 求解器来说，求解速度非常慢，有时候即使缩小仿真步长仍然得不到想要的结果，此时需要设置为刚性方程求解器，如 ode15s 或者 ode23t 等。
- 仿真步长太小。有时在仿真时不需要过小的步长就能够满足计算精度，就可以适当扩大最小步长设置。在满足精度的情况下，扩大步长能够显著地提高仿真速度。
- 误差设置过小。有时在仿真时不需要过小的误差限制就能够满足计算精度，就可以适当扩大误差限制。在满足精度的情况下，扩大误差限制也能够显著地提高仿真速度。
- 在 Simulink 模型中调用了 MATLAB Fcn 模块。当 Simulink 进行仿真时，每次 MATLAB Fcn 模块都要调用相应的 M 函数，使得仿真速度大大的受到影响。在能够利用 Simulink 模块搭建的情况下，尽量使用模块组合来实现 M 函数的功能，可大大加快仿真速度。
- 在 Simulink 模型中调用了 M 文件的 S 函数。和 MATLAB Fcn 模块一样，这个调用功能同样会导致 MATLAB 解释器的介入，降低仿真速度。在能够利用 Simulink 模块搭建的情况下，尽量使用模块组合来实现 S 函数的功能，可大大加快仿真速度。
- 模型中存在有代数环。代数环的存在会大大地降低计算速度，甚至可能导致仿真失败，关于如何消除代数环可参考后面的章节。
- 模型中包含有 Memory 模块。当求解器为 ode15s 或者 ode113 等变阶方法时，仿真速度会受到影响。
- 模型中的积分模块的输入为一个随机信号。
- 模型中有混合系统存在。在混合系统中，采样时间不为整数倍时，往往会导致计算速度下降。

3.2.2 Simulink 提高精度

对于提高 Simulink 的仿真精度，有一个比较笼统的方法就是缩小仿真步长，或者设置较小的相对误差，或者选择合理的求解器，操作步骤如下：

(1) 选择适当的求解器。判断系统是否是刚性的，如果是则选择 ode15s 或者 ode23t 等求解器，如果不是则选择 ode45 等。

(2) 确定误差。首先设置一个相对误差值，然后依次缩小相对误差，看仿真结果是否有明显变化，如果没有表示相对误差设置接近理想范围。绝对误差设置过程大致类似。

(3) 调整仿真步长，方法类似于相对误差的调整。有时候需要将仿真步长与相对误差结合起来调整，需要经过反复数次才能够得到理想的精度。

如果仿真结果不稳定，可能的原因如下：

- 系统本身不稳定，可能出现异常现象。
- 如果使用的求解器为 ode15s，可以将最大阶数定为 2，或者将求解器调整为 ode3s 进行尝试。
- 可能是相对误差或者绝对误差的设置不够理想。

3.2.3 MATLAB 加速计算

Simulink 和 MATLAB 是密不可分的，前面谈了如何提高 Simulink 模型的仿真速度，现在简单地谈谈如何提高 MATLAB 的计算速度。

MATLAB 语言是一种解释性语言，所以有时 MATLAB 程序的执行速度不够理想。这里将依照编者多年的实际编程经验和收集的相关资料，给出加快 MATLAB 程序执行速度的一般性建议。

1. 尽量避免使用循环

循环语句及循环体经常被认为是 MATLAB 编程的瓶颈问题。改进这样的状况有两种方法：

1) 尽量用向量化的运算来代替循环操作。下列将通过例子演示如何将一般的循环结构转换成向量化的语句。

【例 0301】考虑无穷级数求和问题

如果要求出其中前有限项，如 100 000 项之和。要精确地求出级数的和，不需要求 100 000 这么多项，往往前几十项就能得出满意的精度。在此主要是为了演示循环运算向量化的优越性。

常规语句进行计算

```
>> tic
s=0;
for i=1: 500000
    s=s+(1/2^i+1/3^i);
end
s
toc
s =
    1.5000
Elapsed time is 3.375000 seconds.
```

如果采用向量化的方法，则可以得出结果如下：

```
>> tic
i=1: 500000;
s=sum(1./2.^i+1./3.^i)
toc
s =
    1.5000
Elapsed time is 1.956000 seconds.
```

可以看出，采取向量化的方法比常规循环运算效率要高得多，如果在一个较大的程序中有多处类似的地方，就能够明显地改变程序的执行效率。

2) 在必须使用多重循环的情况下，如果这些循环执行的次数不同，则建议将循环次数最少的循环放在最外一层，循环次数越多的越放到靠近最内一层，这可以明显提高仿真

速度。

【例 0302】考虑生成一个 10×10000 的 Hilbert 长方矩阵，该矩阵的定义是其第 i 行第 j 列元素为 h_{i,j}=1/(i+j-1)。

执行代码 1：可以由下面语句先进行 i=1：10 的循环后进行 i=1：10000 的循环。

```
>> tic
for i=1: 10
    for j=1: 10000
        H(i,j)=1/(i+j-1);
    end
end
toc
Elapsed time is 3.266000 seconds..
```

执行代码 2：可以由下面语句先进行 i=1：10000 的循环后进行 i=1：10 的循环。

```
>> tic
for j=1: 10000
    for i=1: 10
        J(i,j)=1/(i+j-1);
    end
end
toc
Elapsed time is 22.797000 seconds.
```

两个程序的耗时区别和前面分析的是一致的。

2. 大型矩阵的预先定维

给大型矩阵动态地定维是个很费时间的事。在定义大矩阵时，首先用 MATLAB 的内在函数，如 zeros() 或 ones() 对变量先进行定维，然后再进行赋值处理，这样会显著减少所需的时间。

【例 0303】

执行代码 1：

```
>> tic
H=zeros(10,10000);
for i=1: 10
    for j=1: 10000
        H(i,j)=1/(i+j-1);
    end
end
toc
Elapsed time is 0.312000 seconds.
```

执行代码 2：

```
>> tic
J=zeros(10,10000);
for j=1: 10000
    for i=1: 10
        J(i,j)=1/(i+j-1);
    end
end
toc
Elapsed time is 0.359000 seconds.
```

可以看出，两个程序的耗时区别和前面分析的是一致的。预先定维后，所需要的时间显著地减少了。同样一个问题，由于采用了有效的措施，执行代码 1 所需的时间就可以从 3.266s 减少到 0.312s，即效率提高了 10 倍多。执行代码 2 所需的时间就可以从 22.797s 减少到 0.359s，即效率提高了 63 倍多。这是用 MATLAB 7.0 执行得到的结果，如果用以前的版本，可能提高的会更多，一般都在百倍以上。

3. 对二重循环这样的特殊问题，还可以使用 meshgrid() 函数构造两个 10x10000 矩阵 i 和 j，从而直接得出 H 矩阵，更进一步地加快速度

【例 0304】

```
>> tic
[i,j]=meshgrid(1: 10,1: 10000);
H=1./(i+j-1);
toc
Elapsed time is 0.031000 seconds.
```

计算所需要的时间进一步减小，同时也表明，MATLAB 的强项就在于矩阵运算。

4. 优先考虑内在函数

矩阵运算应该尽量采用 MATLAB 的内在函数，因为内在函数是由更底层的编程语言 C 语言构造的，其执行速度显然快于使用循环的矩阵运算。

5. 采用更加有效的算法

在实际应用中，解决同样的数学问题经常有各种各样的算法。例如求解定积分的数值解法在 MATLAB 中就提供了两个函数 quad()和 quad8(),其中后一个算法在精度、速度上都明显优于前一种方法。所以说，在科学计算领域是存在“多快好省”的途径的。如果一个方法不能满足要求，可以尝试其他的方法。

6. 应用 Mex 技术

虽然采用了很多措施，但执行速度仍然很慢，比如说耗时的循环是不可避免的，这样就应该考虑用其他语言，如 C 或 Fortran 语言。按照 Mex 技术要求的格式编写相应部分的程序，然后通过编译联接，形成在 MATLAB 可以直接调用的动态连接库(DLL)文件，这样可以显著地加快运算速度。

7. 遵守 Performance Acceleration 的规则

关于“Performance Acceleration”的概念可参阅 matlab 帮助文件，下面将其规则总结如下 7 条。

1) 只有使用以下数据类型，matlab 才会对其加速，如 logical、char、int8、uint8、int16、uint16、int32、uint32、double，而语句中如果使用了非以上的数据类型则不会加速，如 numeric、cell、structure、single、function handle、java classes、user classes、int64、uint64。

2) matlab 不会对超过三维的数组进行加速。

3) 当使用 for 循环时，只有遵守以下规则才会被加速：

- for 循环的范围只用标量值来表示；

- for 循环内部的每一条语句都要满足上面的两条规则，即只使用支持加速的数据类型，只使用三维以下的数组，而且循环内只调用了内建函数(build-in function)。

4)　当使用 if、elseif、while 和 switch 时，其条件测试语句中只使用了标量值时，将加速运行。

5)　不要在同一行中写入多条操作，这样会减慢运行速度。即不要有这样的语句：

```
x = a.name; for k=1: 10000, sin(A(k)),end;
```

6)　当某条操作改变了原来变量的数据类型或形状(大小，维数)时将会减慢运行速度。

7)　应该这样使用复常量 x = 7 + 2i，而不应该这样使用：x = 7 + 2×I，后者会降低运行速度。



第 4 章　Simulink 模块库

熟悉 Simulink 模块库所包含的内容是建立模型的基础，只有熟练地掌握了模块库才能够让用户快速地建立模型，或者以最少的模块来建立模型，或者以最快仿真速度为目的来建立模型。总之，就是能够让我们多、快、好、省地建立模型。

本章将详细地介绍各个模块库的作用，并介绍其中常用的模块。

本章主要内容包括：

- 模块库简介
- 常用模块组 Commonly Used Blocks
- 连续模块组 Continuous
- 离散模块组 Discrete
- 非连续模块组 Discontinuities
- 逻辑运算模块组 Logic and Bit Operations
- 函数与表格模块组
- 数学运算模块组 Math Operations
- 端口与子系统模块组 Ports&Subsystems
- 信号通道模块组 Signal Routing
- 信号接受模块组 Sinks
- 信号源模块组 Sources
- 用户自定义模块组

4.1　模块库简介

在 MATLAB 命令窗口输入“Simulink”或者单击 MATLAB 工具栏中的 Simulink 图标，将会打开 Simulink 模型库浏览器窗口，如图 4.1 所示。或者在 MATLAB 命令窗口中输入“Simulink3”命令，弹出如图 4.2 所示窗口，这是传统 Simulink 模块库显示方式。

从图 4.1 可看出，Simulink 模型库浏览器窗口中包含了丰富的模块组，主要包括 Simulink 基本模块，Aerospace blockset，Fuzzy Logic Toolbox，Real Time Workshop，SimMechanics，SimPowerSystem，Virtual Reality Toolbox 和 Stateflow 等。由于 Simulink 能够与 MATLAB 完美地结合，还能够调用 MATLAB 中的诸多工具箱，具体可以在 MATLAB 命令窗口中输入：

```
>> ver
```

就会在 MATLAB 命令窗口输出 MATLAB/Simulink 所包含的所有工具箱，不难发现这些工具箱涵盖了很多科技领域。

为了方便介绍浏览器中各个 Simulink 模块组，可用另外一种方式访问这些模块组，这样可以更好地显示模块组的全貌。操作方法：右击相应的模块组名，从弹出的菜单中选

择 Open the Continuous library 命令，如图 4.3 所示，这样就会弹出相应的模块组界面。

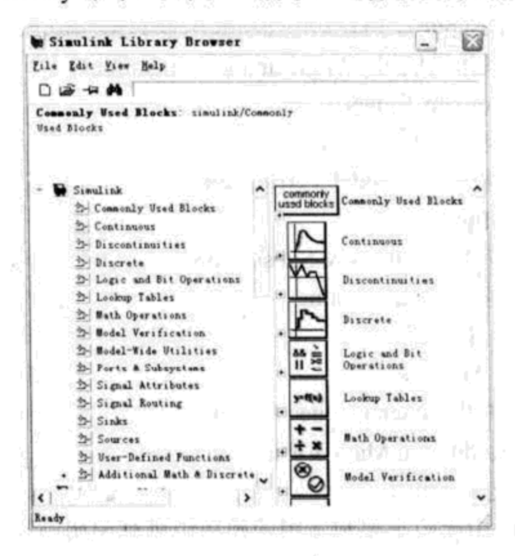


图 4.1 Simulink 模型库窗口

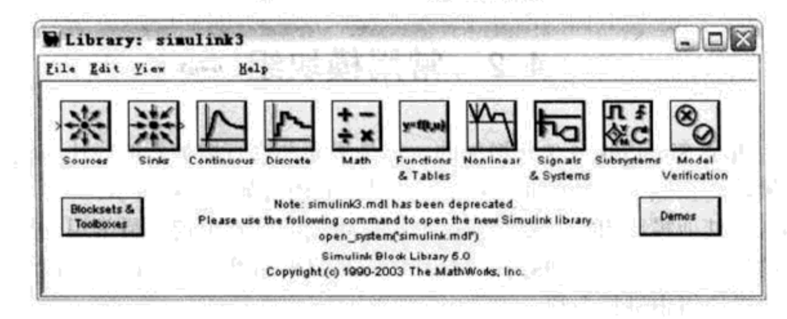


图 4.2 传统 Simulink 模块库显示方式

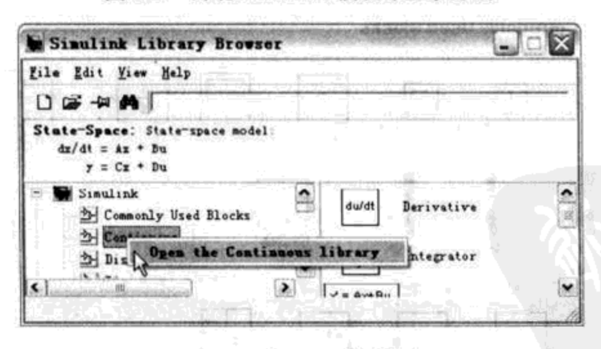


图 4.3 打开具体模块库

从图 4.1 所示的窗口，即模块浏览器左侧可以看出，可以看到模块库可以由各个模块组构成。整个 Simulink 模块库中包含了以下若干模块组：

- 常用模块组 Commonly Used Blocks
- 连续模块组 Continuous

- 非连续模块组　Discontinuities
- 离散模块组　Discrete
- 逻辑与二进制操作模块组　Logic and Bit Operations
- 寻表操作组　Lookup Tables
- 数学操作模块组　Math Operations
- 模型确认操作模块组　Model Verification
- Model-Wide Utilities
- 端口与子系统模块组　Ports & Subsystems
- 信号路由模块组　Signal Routing
- 接受器模块组　Sinks
- 信号源模块组　Sources
- 自定义函数模块组　User-Defined Functions
- 附加操作组　Additional Math & Discrete

此外，用户可以自定义模块组。

本章只对 Simulink 中比较重要和常用的模块组进行说明，其他工具箱部分包含在后面的专门章节，在此不赘述。

4.2　常用模块组

这个模块组是 Simulink 6.0 新增加的，但是里面并没有增加新的模块，模块均为其他模块组中的模块。按照 4.1 节中的显示方法，显示其内容如图 4.4 所示。

模块均来自其他模块组，主要是为了方便用户能够在其中调用最常用的模块，而不必到模块所属的模块组一个一个地寻找，有利于提高建模速度。本模块组中模块的具体使用方法会在其他模块组中进行介绍。

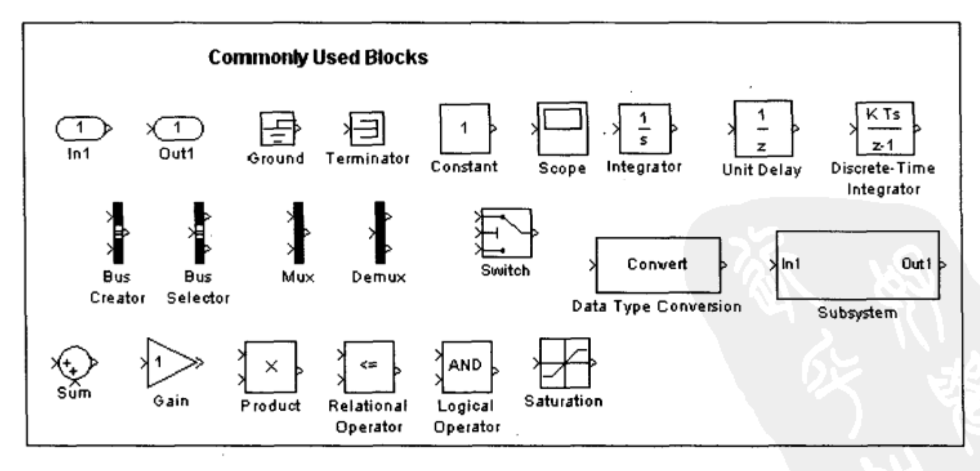


图 4.4　常用模块组

4.3　连续模块组

连续模块组包括常用的连续模块，如图 4.5 所示，包含了连续模型中所涉及的模块。

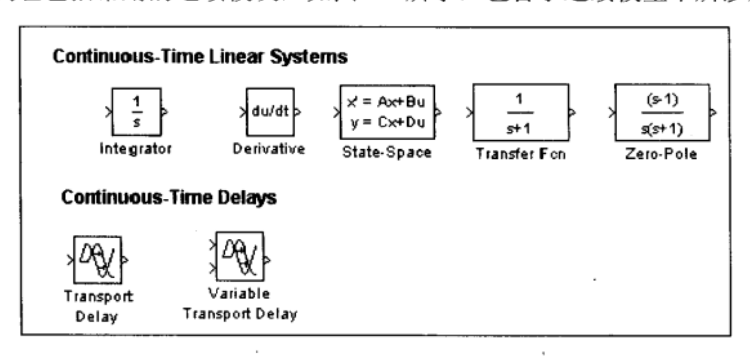


图 4.5　连续模块组

- 积分模块(Integrator)：积分模块将输入信号经过数值积分，在输出端输出相应的结果。建立动力学方程时需用到此模块。
- 微分模块(Derivative)：微分模块将输入信号经过数值微分，在输出端输出相应的结果。此模块一般较少使用，应尽量避免，容易导致较大误差。
- 状态空间模块(State-Space)：表示一种线性系统的时域表示，通称的线性微分方程都可以转变成状态方程形式，表示成如下方程：

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases}$$

  其中：$A \in R^{n \times n}$，$B \in R^{n \times m}$，$C \in R^{l \times n}$，$D \in R^{l \times m}$，u 为输入信号，x 为状态变量，y 为输出信号。
- 传递函数模块(Transfer Fcn)：传递函数通常用在频域分析方面，通过 Laplace 变换，将一个线性微分方程转换成代数形式，形式如下：

$$G(s) = \frac{b_1 s^m + b_2 s^{m-1} + \cdots + b_m s + b_{m+1}}{s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_{n-1} s + a_n}$$

  其中：$m < n$。
- 零极点模块(Pole-Zero)：实现的功能类似于传递函数模块，只是表达形式不同。是对传递函数模块的分子分母进行相应的因式分解，形式如下：

$$G(s) = K\frac{(s+z_1)(s+z_2)\cdots(s+z_m)}{(s+p_1)(s+p_2)\cdots(s+p_n)}$$

  其中：$m < n$，K 为系统的增益。
- 时间延迟模块(Transport Delay)：时间延迟模块将输入信号进行延迟，延迟时间在模块内部进行设置。
- 可变时间延迟模块(Variable Transport Delay)：时间延迟模块将输入信号进行延迟，延迟时间通过第二个端口进行设置。

4.4 离散模块组

离散模块组包括常用的离散模块，如图 4.6 所示。

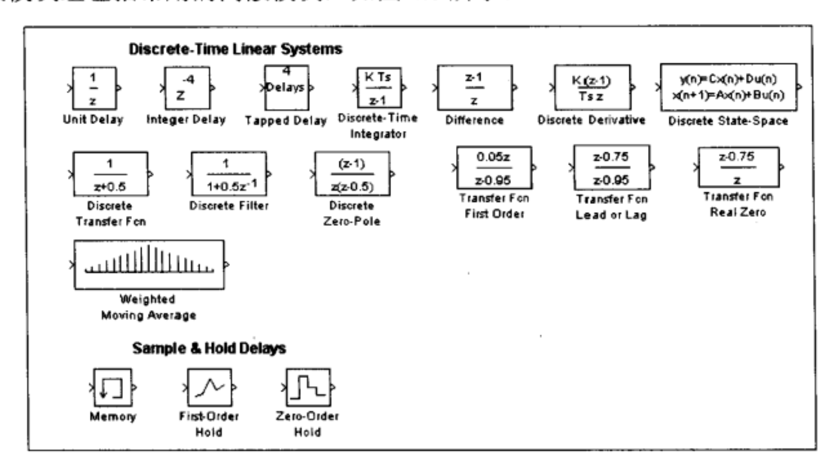


图 4.6 离散模块组

- 单位延迟模块(Unit Delay)：对输入信号进行单位延迟变换，模块输出为单位延迟信号。
- 积分延迟(Integer Delay)：对输入信号进行 N 步信号延迟，输入信号可以是标量，也可以是向量。
- 多步抽头积分延迟模块(Tapped Delay)：对输入信号进行延迟，可以是多步或单步的，输入信号是标量。其余积分延迟的不同在于，如果同样延迟 4 步，多步抽头积分延迟则会出现 4 条延迟曲线，分别是延迟 1 步，2 步，3 步及 4 步，而积分延迟只有一条曲线，只延迟 4 步。
- 离散时间积分模块(Discrete－Time Integrator)：执行离散信号积分，或输出信号累计，可取代积分模块产生纯离散系统。
- 离散传递函数(Discrete Transfer Fcn)，离散滤波器(Discrete Filter)，离散零极点模块(Discrete Zero-Pole)：离散传递函数定义如下：

$$G(z)=\frac{b_1z^m+b_2z^{m-1}+\cdots+b_mz+b_{m+1}}{z^n+a_1z^{n-1}+b_2z^{n-2}+\cdots+a_{n-1}z+a_n}$$

  离散滤波器和离散零极点模块都是其特殊形式和变形形式。
- 差分模块(Difference)：计算一步内信号的变化，就是用输入值减去上一时刻的输入值。
- 记忆模块(Memory)：输出前一时刻的输入。
- 零阶保持器(Zero-Order Hold)：在计算步长内使输出值保持相同。
- 一阶保持器(First-Order Hold)：在计算步长内，按照一阶插值的方法进行插值，然后输出。

其他模块使用比较少，大致都是前面介绍的特殊形式，具体问题可以使用帮助查询，在此不赘述。

4.5 非连续模块组

非连续模块组包括常用的离散模块，如图 4.7 所示。

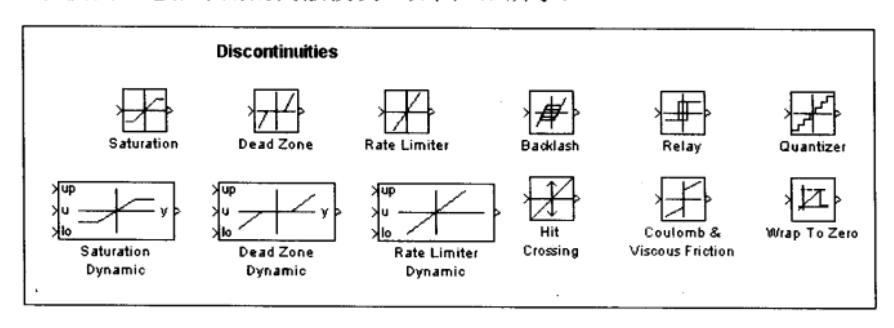


图 4.7 非连续模块组

- 饱和模块(Saturation)：可以设置模块的上下限，当信号超过上限时，用上限代替，当信号小于下限时，用下限代替。
- 死区模块(Dead Zone)：可以设置死区开始和结束的时间，当在此时间段内，输出信号为零。
- 斜率限制模块(Rate Limiter)：线性信号变化的速率，变化的速率不能超过上限，变化的速率不能小于下限。

  变化的速率公式如下：

$$rate = \frac{u(i) - u(i-1)}{t(i) - t(i-1)}$$

  模块输出信号分为三种：
  - ◆ 当速率大于速率上限(R)，模块输出为 $y(i) = \Delta t \times R + y(i-1)$
  - ◆ 当速率小于速率上限(F)，模块输出为 $y(i) = \Delta t \times F + y(i-1)$
  - ◆ 当速率介于速率上、下限之间，模块输出为 $y(i) = u(i)$
- 动态饱和模块(Saturation Dynamic)：类似于饱和模块，只是上下限可以由外部信号确定。
- 动态死区模块(Dead Zone Dynamic)：类似于死区模块，只是上下限可以由外部信号确定。
- 动态斜率限制模块(Rate Limiter Dynamic)：类似于斜率限制模块，只是上下限可以由外部信号确定。

4.6 逻辑运算模块组

逻辑模块组包括常用的离散模块，如图 4.8 所示。

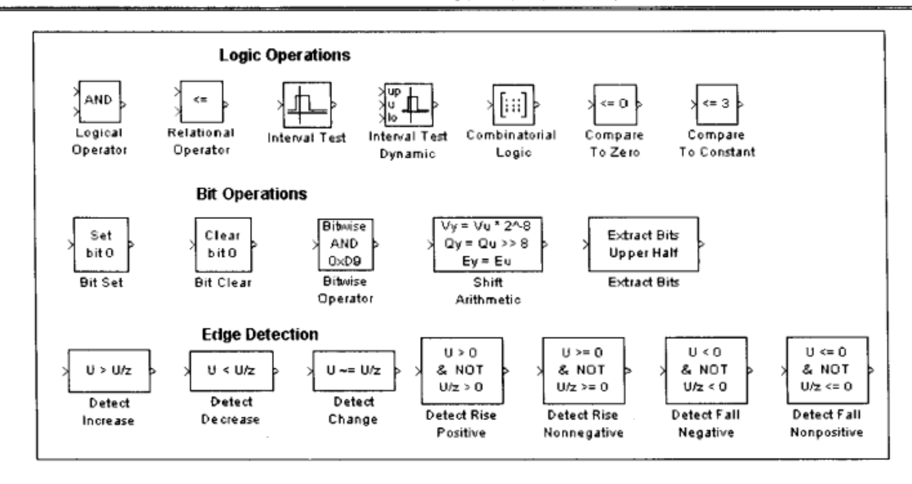


图 4.8 逻辑模块组

- 逻辑运算模块(Logical Operator)：进行各种逻辑运算及其功能如表 4.1 所示。

表 4.1 逻辑运算及其功能

逻辑运算	功 能
AND	输入全部为真时输出为真
OR	输入有一个为真时输出为真
NAND	输入有一个为非时输出为真
NOR	输入全部为非时输出为真
XOR	输入中有奇数个输入为真时输出为真
NOT	输入为非时输出为真

- 关系运算符(Relational Operator)：判断两个输入端的大小关系，见表 4.2 所示。

表 4.2 关系运算符及其功能

关系运算符	功 能
==	第一个输入等于第二个输入时为真
~=	第一个输入不等于第二个输入时为真
<	第一个输入小于第二个输入时为真
<=	第一个输入小于或等于第二个输入时为真
>=	第一个输入大于或等于第二个输入时为真
>	第一个输入大于第二个输入时为真

- 区间测试模块(Interval Test)和动态区间测试模块(Interval Test Dynamic)：都是判断输入是否属于某个区间。如果是，则输出为真。不同的是，前者的上下界是固定的，后者则是动态输入的。
- 组合逻辑模块(Combinatorial Logic)、较零模块(Compare to Zero)、比较常数模块(Compare to Constant)都是前面几种情况的组合和特殊情况。

- 二进制置 1 模块(Bit Set)：将指定的二进制数设置为 1。
- 二进制清零模块(Bit Clear)：将指定的二进制数设置为 0。
- 二进制逻辑运算(Bitwise Operator)：将输入的二进制与给定的二进制数的对应位置进行逻辑运算，或设定输入端口数量，使得多个端口输入二进制数与给定的二进制数对应位置进行逻辑运算。
- 移位运算(Shift Arithmetic)：将给定的二进制数进行移位，即向左移动或向右移动，具体可以见表 4.3 的例子。或者是将二进制数中的小数点进行移位运算，具体可以见表 4.4 的例子。

表 4.3 二进制数移位运算

移位运算	二进制数	十进制数
没有移位(原始数据)	11001.011	-6.625
二进制数向右移动两位，前面补 1	11110.010	-1.75
二进制数向左移动两位，前面补 0	00101.100	5.5

表 4.4 二进制小数点移位运算

移位运算	二进制数	十进制数
没有移位(原始数据)	11001.011	-6.625
二进制小数点向右移动两位	1100101.1	-26.5
二进制小数点向左移动两位	110.01011	-1.65625

- 二进制数截取模块(Extract Bits)：通过参数设置，截取想要的部分，双击模块，弹出图 4.8 所示对话框，共有 5 种运算。运算结果如表 4.5 所示。

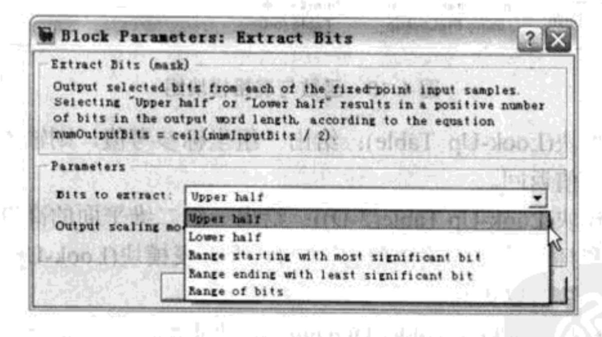


图 4.9 二进制数截取参数设置

表 4.5 截取运算

截取运算	结 果
未进行运算(原始数据)	110111001
Upper half(截取上半部分)	11011
Lower half(截取下半部分)	11001
Range starting with most significant bit(从开头截取期望的位数，如果为 8)	11011100

续表

截取运算	结　果
Range end with most significant bit(从结尾截取期望的位数，如果为 8)	10111001
Range of bits(截取期望的位置，如果为［47］)	1011

- 检测减小模块(Detect Decrease)、检测增加模块(Detect Increase)、检测变化模块(Detect Change)：将输入二进制数与前一个输入进行比较。三者分别对应为：当减小时输出为真，当增加时输出为真，当变化时输出为真。
- Detect Rise Positive，Detect Rise Nonnegative，Detect Fall Negative，Detect Fall Nonpositive 这些模块都可以从模块上的表示看出具体的功能，就不赘述了。

4.7　函数与表格模块组

函数与表格模块组包括常用的离散模块，如图 4.10 所示。

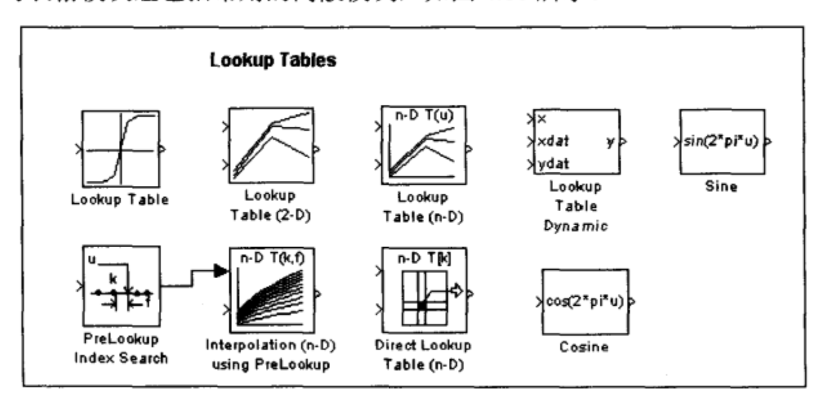


图 4.10　函数与表格模块组

- 一维查表模块(Look-Up Table)：给出一组坐标参考值，则输入量经过查表和插值计算出输出值返回。
- 二维查表模块(Look-Up Table(2-D))：给出一组二维平面的高度值，则输入两个量经过查表和插值计算出输出值返回。N 维查表模块(Look-Up Table(N-D))功能类似，只是维数可以更高。
- 动态查表模块(Lookup Table Dynamic)：功能类似于上面介绍的模块，只是查表的参考数据是动态输入的。
- 直接查表模块(Direct Lookup Table(n-D))：这是通过输入元素所在的位置，也就是索引，来输出相应的结果，结果可以是标量，也可以是矢量。

4.8　数学运算模块组

数学运算模块组包括常用的离散模块，如图 4.11 所示。

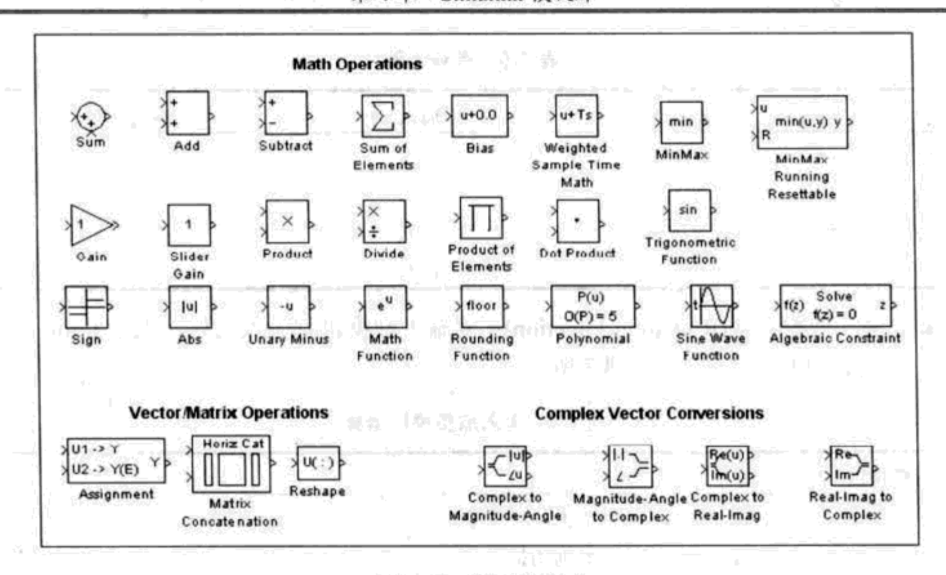


图 4.11　数学模块组

- 求和模块(Sum)、相加模块(Add)、相减模块(Substract)、元素求和模块(Sum of Elements)：这 4 个模块功能类似，均可通过设置达到相同效果，可以改变输入端口数量，对输入进行相加或相减。图形形状均可在圆形和方形间相互转换。后三者为前者的执行形式，功能有重复的嫌疑。
- 偏置模块(Bias)：在输入数据的基础上加上一个偏置常数，然后输出。此功能可通过求和模块和常数模块组合实现。
- 增益模块(Gain)与滑块增益模块(Slider Gain)：在输入信号基础上乘以一个设定的数据，然后输出。不同的是，后者是通过设置滑块，然后可移动滑块来设定增益，参数对话框如图 4.12 所示。

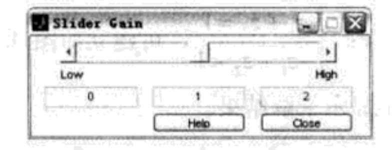


图 4.12　滑块增益模块设置

- 叉乘模块(Product)和叉除模块(Divide)、元素相乘(Product of Elements)：相对数学运算中的叉乘和叉除运算，对输入数据进行相乘和相除。
- 点乘模块(Dot Product)：对输入数据进行相乘，与叉乘模块的不同在于对复数的运算，具体的运算如表 4.6 所示。
- 符号模块(Sign)、绝对值模块(Abs)和取反模块(Unary Minus)：分别求取输入信号的符号，绝对值和取反运算。
- 数学函数模块(Math Function)：对输入信号进行各种设定的输入运算，这些设定的数学运算包括 exp，log，10^u，log10，magnitude^2，square，sqrt，pow，conj，reciprocal，hypot，rem，mod，transpose，hermitian 模块。

表 4.6 点乘运算

实例 1	输入 1	输入 2	输出
实例 1	实数	虚数	虚数
实例 2	实数	实数	实数
实例 3	实数	虚数	虚数
实例 4	虚数	虚数	虚数

- 舍入取整模块(Rounding Function)：对输入数据进行舍入操作，包括 floor、ceil、round 和 fix，用法如表 4.7 所示。

表 4.7 舍入取整模块运算

舍入取整模块	功　能
floor	输出为小于输入数据的最大整数
ceil	输出为大于输入数据的最小整数
round	输出为最接近输入数据的整数
fix	输出为输入数据上下界两个整数中最接近 0 的整数

- 多项式运算模块(Polynomial)：对输入数据进行多项式运算，其中参数为多项式高次项到低次项的系数。
- 取大取小模块(MinMax)：对输入信号取最大值和最小值。
- 三角函数模块(Trigonometric)：进行各种三角函数运算，包括正弦、余弦、正切等。
- 正弦波模块(Sine Wave)：可以设置幅值、相角和频率。
- 代数约束模块(Algebraic Constraint)：对系统进行代数约束，可避免代数环的影响。

【例 0401】对于约束方程：$\begin{cases} z_1 + z_2 = 1 \\ -z_1 + z_2 = 1 \end{cases}$，可建立如图 4.13 所示模型图进行求解，结果可以从 Display 和 Display1 显示求解结果。

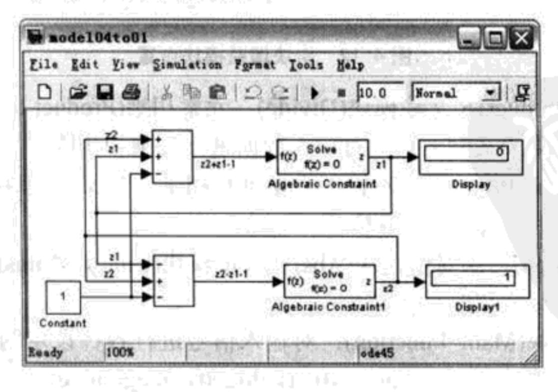


图 4.13 方程求解模型图

- 矩阵串联模块(Matrix Concatenation)：对输入数据进行串联，有两种方式，即水平串联和竖直串联。这两种运算方式如表 4.8 所示。

表 4.8 矩阵串联方式

矩阵串联方式	等价 MATLAB 代码
水平方式(Horizontal,)	y = [u1 u2 u3…un]
竖直方式(Vertical)	y = [u1;u2;u3;…;un]

说明： 表中 n 在模块参数对话框中设置。

- 修改矩阵维数模块(Reshape)：修改输入矩阵的维数，输出分别为 1-Darray，Column vector，Row vector，Customize。说明见表 4.9 所示。

表 4.9 输出方式

输出方式	功 能
1-Darray	将二维矩阵转换成一维矩阵，输出矩阵为输入矩阵第一列，紧接第二列，依此类推。如果输入的是向量，将不会改变
Column vector	将输入矩阵和向量转换成列矩阵，即为 M×1 维矩阵，M 为输入数据的元素个数
Row vector	将输入矩阵和向量转换成行矩阵，即为 1×N 维矩阵，N 为输入数据的元素个数
Customize	将输入矩阵和向量转换成[M N]矩阵，但是输入数据必须匹配矩阵的元素个数

- 复数转换成幅相表示模块(Complex to Magnitude-Angler)和幅相转换成复数表示模块(Magnitude-Angle to Complex)：前者将输入复数转换成幅值和相角的表示方法，后者相反，将幅值和相角表示的数据转换成复数表示方法。
- 输出复数的实部和虚部模块(Complex to Real-Imag)和将实部和虚部转换成复数模块(Real-Imag to Complex)：前者将输入的复数的实部和虚部分开输出，后者将输入的数据组合成复数。

4.9 端口与子系统模块组

端口与子系统模块组包括常用的离散模块，如图 4.14 所示。

- 空白子系统(Subsystem，Atomic Subsystem，CodeReuse Subsystem)：搭建子系统模块，给出输入和输出端，允许用户在其间绘制所需的子系统模型。
- 触发子系统(Triggered Subsystem)，使能子系统(Enabled Subsystem)，触发和使能子系统(Enabled and Triggered Subsystem)：分别在触发信号发生时，使能信号发生时，以及使能和触发信号同时发生时，子系统可以工作，这些触发和使能信号都可以自定义。
- 结构控制子系统(For Iterator Subsystem，While Iterator Subsystem)：都是程序控制子系统。
- 转向控制子系统(If Action Subsystem，Switch Case Action Subsystem)：都是通过

条件判断，转向制定的子系统。

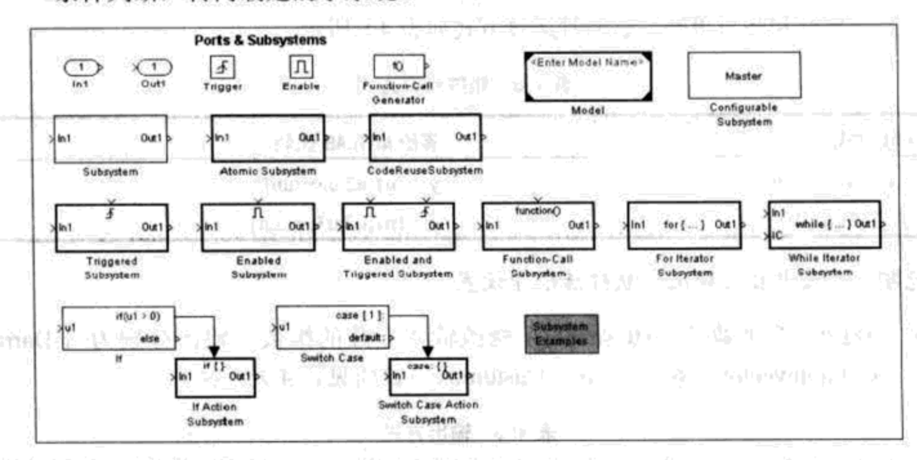


图 4.14 端口与子系统模块组

4.10 信号通道模块组

信号通道模块组包括常用的离散模块，如图 4.15 所示。.

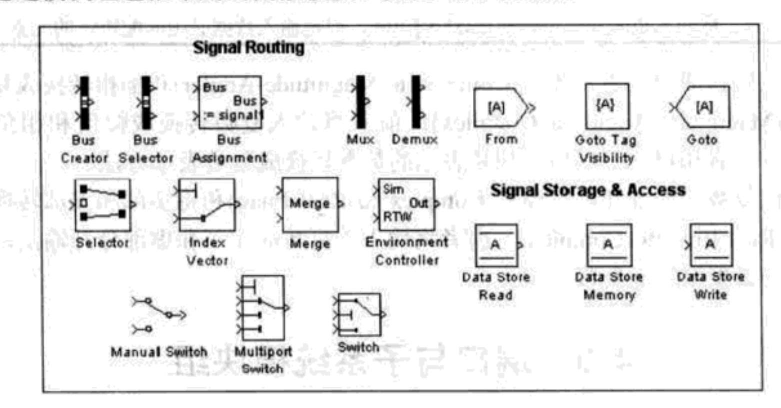


图 4.15 信号通道模块组

- 混路器(Mux)和分路器(Demux)：前者将多路信号依照向量的形式混合成一路信号。后者将前者混合成的一路信号依照原来的顺序分解成多路信号。
- 手工转换器(Manual Switch)：按要求手工转换连接通路。
- 切换模块(Switch)：通过第二个端口设置限制，在第一个和第三个端口间转换。
- 多端口切换模块(Multiport Switch)：第一个端口是控制端口，其余的是数据端口。通过第一个端口来选择要输出的输入端口。
- Data Store Read，Data Store Memory 和 Data Store Write：是对数据进行读取、存储和写入到内存的模块。

4.11　信号接受模块组

信号接受模块组包括常用的离散模块，如图 4.16 所示。

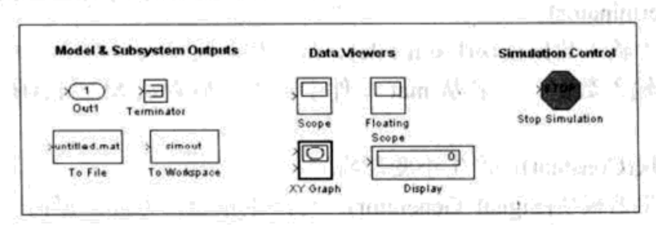


图 4.16　信号接受模块组

- 输出到工作空间模块(Out1)：用来反映整个系统的输出端，这样的设置在模型线性化与命令行仿真时是必需的，在系统直接仿真时这样的输出将自动在 MATLAB 工作空间中生成变量。
- 终结模块(Terminator)：用来终结输出信号，在仿真的时候可以避免由于某些模块的输出端无连接而导致的警告。
- 输出数据到文件模块(To File)：将模块输入的数据输出到.mat 文件当中。
- 输出数据到工作空间模块(To Workspace)：将模块输入的数据输出到工作空间当中。
- 示波器模块(Scope)：将输入信号输入到示波器中显示出来。
- X-Y 示波器模块(XY Graph)：将两路信号分别作为示波器的两个坐标轴，以显示信号的相轨迹。
- 显示数据模块(Display)：将输入信号以数字形式显示出来。
- 终止仿真模块(Stop Simulation)：如果输入为非零，则强制终止仿真。

4.12　信号源模块组

信号源模块组包括常用的离散模块，如图 4.17 所示。

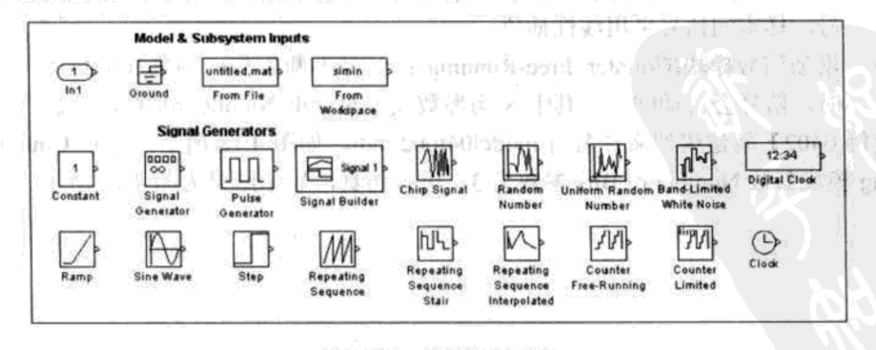


图 4.17　信号源模块组

- 输入端口模块(In1)：用来反映整个系统的输入端，在模型线性化与命令行仿真时，这个设置非常有用，可作为信号输入。
- 接地模块(Ground)：一般用于表示零输入模块，如果一个模块的输入端没有接任何其他模块，仿真时往往会出现警告，这样可以将该模块接入，功能类似与终结模块(Terminator)。
- 从文件中输入数据模块(From File)、从工作空间输入数据模块(From Workshop)：从外部输入数据，前者从.mat 文件中输入，后者从 MATLAB 工作空间中输入数据。
- 常数模块(Constant)：产生不变常数。
- 信号发生器模块(Signal Generator)：可产生正弦，方波，锯齿等信号，并可设置幅值和频率。
- 脉冲发生器模块(Pulse Generator)：产生脉冲信号，可以设置幅值，周期，宽度等参数。
- 信号构造模块 (Signal Builder)：在模块窗口双击此模块，在弹出对话框中绘制信号。
- 斜坡信号模块(Ramp)、正弦波信号模块(Sine Wave)、阶跃信号模块(Step)：分别产生斜坡，正弦和阶跃信号。
- 重复信号模块(Repeating Sequence)：构造可重复的输入信号。
- 变频信号模块(Chirp Signal)：输出变频的正弦信号。
- 随机信号模块(Random Number)和均匀分布随机信号模块(Uniform Random Number)：都是产生随机信号，不同的是前者为正态分布随机信号，而后者为均匀分布随机信号。
- 带宽限制白噪声(Band-Limited White Noise)：一般用于连续或混杂系统的白噪声信号输入。
- 时钟模块(Clock)和数字时钟模块(Digital Clock)：前者用于显示和提供仿真时间信号，后者则是在指定的样本间隔内显示时间，其他情况均保持时间不变。
- 重复离散信号模块(Repeating Sequence Stair)：构造可重复的输入离散信号，样本间信号采用零阶保持。
- 重复离散信号模块(Repeating Sequence Interpolated)：构造可重复的输入离散信号，样本间信号采用线性插值。
- 累加信号模块(Counter Free-Running)：信号不断累加，当累加的信号大于 2^N-1 时，信号会自动回零，其中 N 为参数设置对话框 Number of Bits 所设置。

【例 0402】新建模型文件名为 model04to02.mdl，如图 4.18 所示，设置 Counter Free-Running 模块参数 Number of Bits 参数为 3。进行仿真，显示结果为图 4.19 所示。

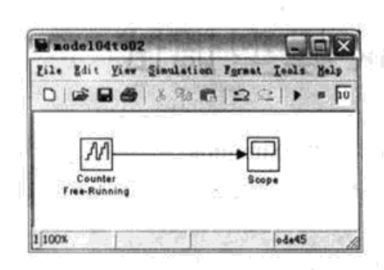


图 4.18　文件 model04to02.mdl 模型

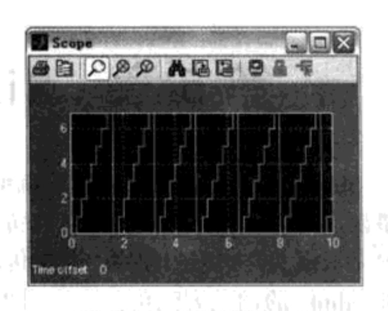


图 4.19　模型仿真结果

4.13　用户自定义模块组

打开用户自定义模块组，包括常用的离散模块，显示内容如图 4.20 所示。

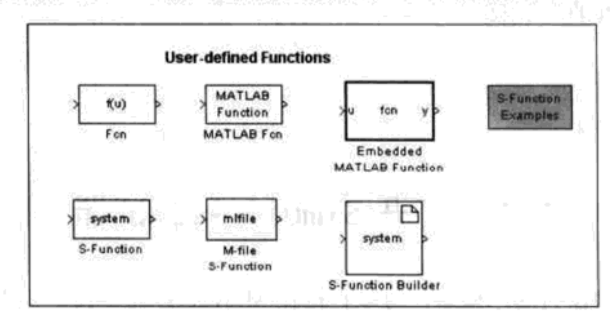


图 4.20　用户自定义模块组

- 函数组合模块(Fcn)、MATLAB 函数模块(MATLAB Fcn)、S 函数模块(S-Function)和 M 文件 S 函数(M-file S-Function)：将各种 MATLAB 函数进行组合，参数为模块输入并直接调用 MATLAB 函数、调用 S 函数和调用 M 文件形式的 S 函数。
- 嵌入式 MATLAB 函数模块(Embedded MATLAB Function)：调用自己编写的 M 文件，与 M 文件 S 函数模块不同的是，在模型窗口双击此模块会弹出 M 文件编辑框，然后就可以自行随意的编写能够完成期望功能的代码。
- S 函数构造模块(S-Function Builder)：从你所提供的 C 代码中创建一个 S 函数。

第5章 Simulink 模型调试

当 Simulink 模型建立好之后，就可以进行仿真了。但 Simulink 可能存在这样或者那样的错误，用户可以一步一步地运行仿真，以便发现模型问题所在。在模型调试过程中，实时地显示模型的状态和模块的数据传输。

Simulink 的调试方法可分为窗口界面调试和命令调试方式。窗口调试适合于初级用户，能够完成绝大多数情况下的调试任务。命令调试方式能够让用户随心所欲地显示调试中的任何信息，包括窗口调试方式不能完成的调试任务，对一个 Simulink 高级用户来说，掌握这种方式是非常有必要的。

本章主要内容包括：

- 打开 Simulink 调试器
- 进行模型仿真与调试
- 断点设置
- 显示仿真信息
- 显示模型信息

5.1 打开 Simulink 调试器

本章以 vdp 模型作为演示模型，可以直接在 MATLAB 命令窗口输入：

```
>> vdp
```

则会自动弹出 vdp 模型的 Simulink 窗口。

5.1.1 窗口调试方式

在 Simulink 模型模型窗口中单击按钮，或者选择 Tools | Debugger 命令，就会启动 Simulink 调试器，如图 5.1 所示。

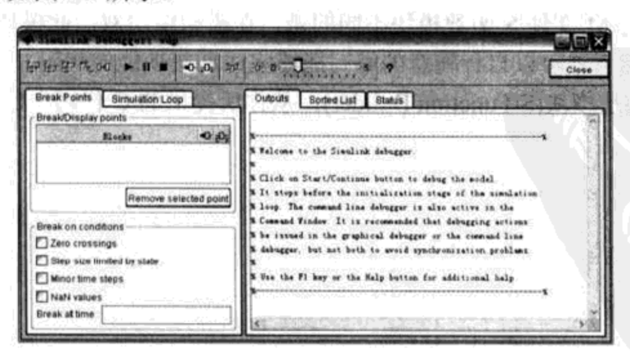


图 5.1 Simulink 调试器

在此约定将整个 Simulink 调试窗口分成 3 个部分：窗口上部的快捷按钮、左边的控制选项框以及右边的信息显示框，分别介绍如下：

1. 快捷按钮

窗口上部的快捷按钮主要是一些经常使用的调试命令，具体如下：

- 运行到下一个模块。
- 开始或者继续运行仿真。
- 终止调试过程。
- 在被选择的模型前设置断点。
- 在被选择的模块处设置显示点，在运行仿真时显示它的输入输出。
- 显示被选择模块当前时间步的输入输出。
- 进入当前方法。
- 跨过当前方法。
- 暂时离开当前方法。
- 在下一个时间步返回第一个方法。
- 开启或者关闭动画。
- 0 ……… 5 动画延迟。
- ? 帮助。

说明： 此处的方法表示 Simulink 仿真开始时的一个最底层动作，一个模块或许会包括很多方法，这个要涉及到模块的函数编写，如模块函数中的一个加、减等。

2. 控制选项

在左边的控制选项框中有四个复选框和一个文本框，主要用于条件断点的设置。

- Zero crossing：模型在遇到过零点检测时产生断点。
- Step size limited state：在状态受到条件约束时产生断点。
- Minor time steps：仿真进入最小时间步模式。
- NaN value：仿真过程中遇到无限大，或者超过机器的最大表示范围，或者遇到一个非数时产生断点。
- Break at time：设置具体的时间以产生断点。

3. 显示信息

右边信息显示框含有 3 个标签：Outputs，Execution Order 和 Status。

- Outputs：用来显示调试后的输出结果，其内容与 MATLAB 命令窗口中的显示内容相同。
- Execution Order：用来显示模块的进程顺序表。
- Status：显示调试的当前设置，它的作用与直接在 MATLAB 命令窗口中输入命令“status”的结果一样。

5.1.2 命令行调试

除了窗口调试方式外，还可以通过 MATLAB 命令窗口进行调试，这对于 MATLAB 高级用户来说显得更加方便快捷。

1. 启动调试器

可以通过两种方法来启动 Simulink 调试器，分别是 sim 命令和 sldebug 命令。例如启动 vdp 模型调试器方法为：

```
>> sim('vdp',[0,10],simset('debug','on'))
```

或者

```
>> sldebug 'vdp'
```

这两个命令先把 vdp 模型加载到内存中，弹出如图 5.2 所示模型图，并在 MATLAB 命令窗口显示开始仿真的时间和调试命令，命令提示符显示模块的索引和将要被执行的第一个模块的名称，具体如下一些信息：

```
%----------------------------------------------------------------------%
[TM = 0                    ] vdp.Simulate
(sldebug @0): >>
```

此时用户可以在提示符后面输入调试命令或者其他的任何 MATLAB 命令进行操作。如果需要结束仿真调试可输入 stop 命令，如果需要获得调试命令的帮助信息，可以输入 help 命令。

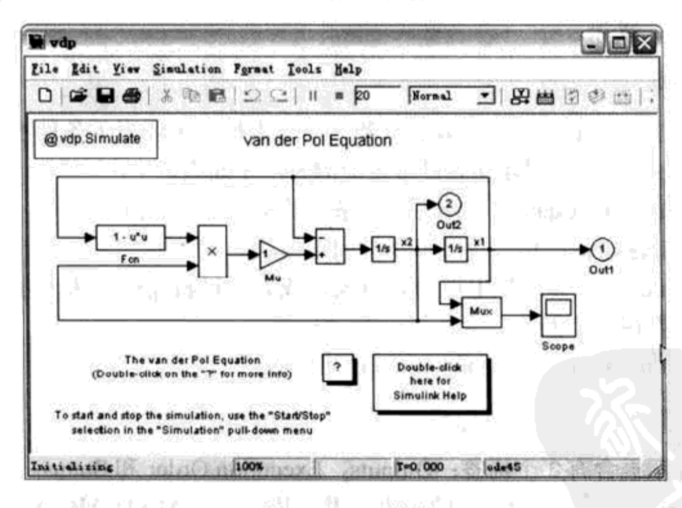


图 5.2 vdp 模型

2. 索引模块

Simulink 是通过模块索引来制定模块的，具体格式为 s:b，其中 s 为模型中当前模块的系统的标号，b 表示是系统中的第几个模块。由于虚拟模块对模块的执行没有任何作用，在调用时不会为它们制定索引。例如 0:2 表示模型系统 0 的第 2 个模块。如果要知道

模型中各个模块的索引，可以在 MATLAB 命令窗口中输入命令“slist”来显示，如：

```
(sldebug @0): >> slist
---- Sorted list for 'vdp' [9 nonvirtual blocks, directFeed=0]
  0:0    'vdp/x1' (Integrator)
  0:1    'vdp/Out1' (Outport)
  0:2    'vdp/x2' (Integrator)
  0:3    'vdp/Out2' (Outport)
  0:4    'vdp/Scope' (Scope)
  0:5    'vdp/Fcn' (Fcn)
  0:6    'vdp/Product' (Product)
  0:7    'vdp/Mu' (Gain)
  0:8    'vdp/Sum' (Sum)
(sldebug @0): >>
```

3. 与 MATLAB 进行交互式工作

在调试命令运行模式下仍然可以使用 MATLAB 命令进行各种操作，可以实现与 MATLAB 之间的交互式工作，例如在仿真模型中将输出和时间保存到变量 yout 和 tout 中，可以在 MATLAB 中直接调用 plot 命令来绘制结果曲线如下：

```
(sldebug @0): >> plot(tout,yout)
```

5.2 进行模型仿真与调试

打开模型后，选择 Tools | Simulink Debugger 命令就可以进行模型调试了，可以在如图 5.1 所示调试窗口单击【开始/继续】按钮 ▶ 进行模型调试。利用 Simulink 模型窗口进行调试，在 MATLAB 命令窗口同时会出现相应的命令调试信息，但是应该尽量避免在 MATLAB 命令行中调试，以防止由于 Simulink 模型窗口与 MATLAB 命令窗口之间异步而导致的错误。如果要用 MATLAB 命令窗口进行，则按 5.1.2 小节中的方法进行操作。

Simulink 允许用户从一个模块执行到另外一个模块，从一个时间点执行到另外一个时间点，或者从一个端点执行到下一个端点，这些操作可通过几个简单的命令来实现，即所谓的单步调试命令。

表 5.1 单步执行调试器命令

命　令	功　能
Step	推进到下一个模块
Next	推进到下一个时间步
Continue	推进到下一个断点
Run	直接推进到仿真结束，不考虑中间存在的断点

1. 单时间步内模块单步执行

在仿真过程中如果需要单步执行，可以单击 ⊡ 按钮。如果在 MATLAB 命令窗口运行时，可以输入单步命令“step”。当执行此命令后，Simulink 的模型窗口有相应的变化，如图 5.3 所示。比较图 5.2 和图 5.3，会发现 vdp 模型会有一个箭头指向正在执行的模块，在箭头末端显示相应的模块名称。

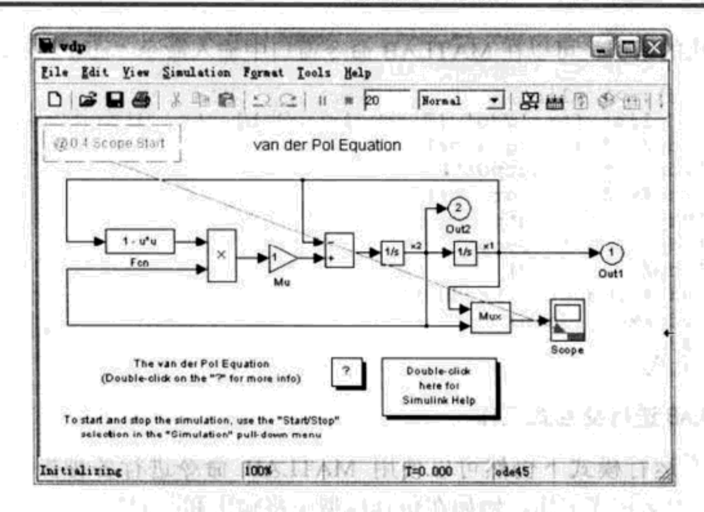


图 5.3 执行单步调试后的 vdp 模型窗口

采用 MATLAB 命令仿真操作如下，只需要输入 step 就可以了。

```
>> sldebug 'vdp'
%----------------------------------------------------------------%
[TM = 0                    ] vdp.Simulate
(sldebug @0): >> step
%----------------------------------------------------------------%
[TM = 0                    ] vdp.Start
(sldebug @1): >> step
%----------------------------------------------------------------%
[TM = 0                    ] RootSystem.Start 'vdp'
(sldebug @2): >> step
%----------------------------------------------------------------%
[TM = 0                    ] 0:4 Scope.Start 'vdp/Scope'
(sldebug @3): >>
```

在执行单步命令以后，Simulink 调试窗口的信息显示框中会出现相应的信息，如图 5.4 所示。

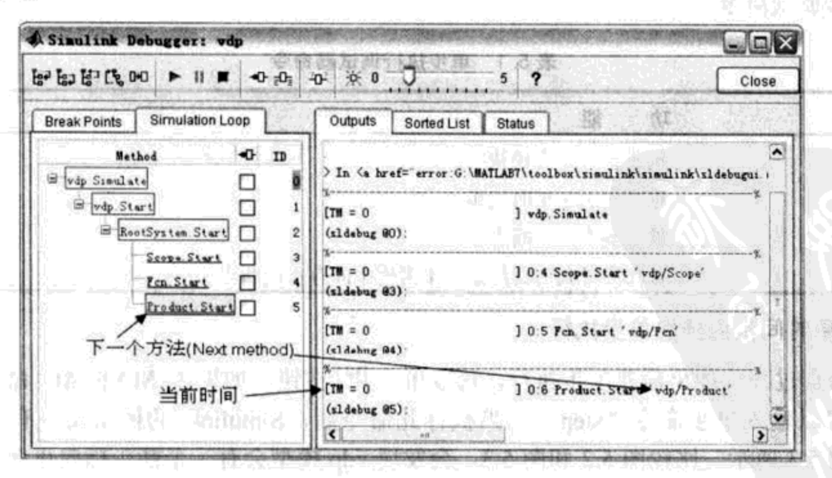


图 5.4 模型 Simulink 调试窗口

2. 按时间步单步执行

与单时间步内模块单步执行不同，这是按时间进程对模型进行调试。当用户执行完模型排序表中的最后一个模块时，调试器就会将模型推进到下一个仿真步长内，并暂停在下一时刻模型排序表中第一个模块的开始处。

在 MATLAB 命令调试方式中，利用 next 命令可以执行当前仿真时间步中的剩余模块，允许用户通过一个命令直接跳到下一个仿真时间步，功能等效于 step over 命令。而 step 是一个模块接一个模块执行，当模型比较复杂时，这个命令的效率太低。

通过下面这段命令执行过程可以看出 step 命令与 next 命令的区别。

```
>> sldebug 'vdp'
%----------------------------------------------------------------%
[TM = 0                    ] vdp.Simulate
(sldebug @0): >> step
%----------------------------------------------------------------%
[TM = 0                    ] vdp.Start
(sldebug @1): >> step
%----------------------------------------------------------------%
[TM = 0                    ] RootSystem.Start 'vdp'
(sldebug @2): >> next
%----------------------------------------------------------------%
[TM = 0                    ] vdp.Initialize.InvariantConstants
(sldebug @8): >> next
%----------------------------------------------------------------%
[TM = 0                    ] vdp.Enable.InvariantConstants
(sldebug @10): >> step
%中间省略部分内容%
%----------------------------------------------------------------%
[Tm = 0                    ] vdp.ode45.IntegrationLoop
(sldebug @43): >> step
%----------------------------------------------------------------%
[Tm = 0                    ] vdp.ode45.EstimateX(t+h*A[0])
(sldebug @44): >> step
%----------------------------------------------------------------%
[Tm = 0                    ] vdp.ode45.ComputeF1
(sldebug @45): >> next
%----------------------------------------------------------------%
[Tm = 2.009509145207664e-005 ] vdp.ode45.EstimateX(t+h*A[1])
(sldebug @61): >> next
%----------------------------------------------------------------%
[Tm = 2.009509145207664e-005 ] vdp.ode45.ComputeF2
(sldebug @62): >> stop
%----------------------------------------------------------------%
% Simulation stopped
```

用 Simulink 窗口调试器时可单击按钮，在调试窗口的右边信息显示框中有类似于 MATLAB 命令窗口中的信息，如下所示。

```
%----------------------------------------------------------------%
[TM = 0                    ] vdp.Simulate
(sldebug @0):
%----------------------------------------------------------------%
[TM = 0                    ] vdp.Start
(sldebug @1):
%----------------------------------------------------------------%
[TM = 0                    ] vdp.Initialize.InvariantConstants
(sldebug @8):
```

3. 运行到下一个断点

continue 命令将仿真从当前的断点运行到下一个断点或者仿真的终点。如果断点先出现，则运行到断点，如果终点先出现，则运行到终点。如果对调试模型中的断点没有任何兴趣，则可以用 run 命令来代替 continue 命令，直接运行到终点而忽略断点的存在。

vdp 模型中存在一个断点，从下列信息中可以看出 continue 的作用。

```
>> sldebug 'vdp'
%----------------------------------------------------------------%
[TM = 0                    ] vdp.Simulate
(sldebug @0): >> continue
%----------------------------------------------------------------%
[TM = 20                   ] vdp.Simulate
(sldebug @0): >> continue
```

5.3 断点设置

知道程序中或者在某种条件下会出现某些问题，此时设置断点来诊断非常有用。定义了断点之后可以借助 continue 命令运行，就可直接找到仿真过程中存在的错误。

作为调试器，设置断点功能是不可少的，Simulink 工具也是一样。对于 Simulink 来说，有两种类型的断点设置方法：无条件断点和有条件断点。

- 无条件断点：不考虑任何条件，只要仿真到达断点就会暂停。
- 有条件断点：在满足条件的情况下，仿真到达断点就会暂停。

关于断点的设置有若干命令，具体如表 5.2 所示。

表 5.2　设置断点命令

命　令	功　能
break	在指定模块前设置断点
bafter	在指定模块后设置断点
ebreak	在求解错误的地方设置断点
tbreak	清除或者设置一个时间断点
xbreak	当仿真步长超出步长限制时
zcbreak	发生过零点时间之前
nanbreak	在数值溢出或者无穷大时设置或者消除断点
rbreak	当仿真需要重新设置求解器时

5.3.1 无条件中断

有 3 种方法可用来设置无条件断点：

- 通过 Simulink 调试器快捷按钮设置断点。
- 通过 Simulink 调试器的 Simulation Loop 面板设置断点。
- 通过 MATLAB 命令窗口设置断点。

下面分别介绍这几种方法。

1. 通过 Simulink 调试器快捷按钮设置断点

首先选中需要设置断点的模块，然后单击调试器快捷按钮中的 Breakpoint 按钮，如图 5.5 所示。

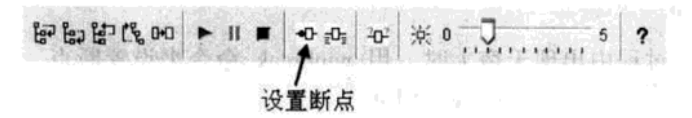


图 5.5 通过 Simulink 调试器快捷按钮设置断点

假如在模型 vdp.mdl 中的模块 Mu 模块前设置断点，可以点选中模块 Mu，然后单击调试器快捷按钮，如图 5.5 所示，然后就会在 Break Points 面板中出现设置断点的模块，如图 5.6 所示。

可以通过非选图 5.6 中 vdp/Mu 模块后边的断点复选框来临时取消断点。如果要删除模块的断点，可以首先选中图 5.6 中的 vdp/Mu 模块，然后单击 Remove selected point 按钮。

2. 通过 Simulink 调试器的 Simulation Loop 面板设置断点

单击 Simulation Loop 面板，选择需要设置断点的模块，使其后的复选框是选中状态如图 5.7 所示。如果要删除断点，则不选中断点模块即可。

3. 通过 MATLAB 命令窗口设置断点

通过 MATLAB 命令窗口设置断点，只适用于命令调试模式。在 MATLAB 命令调试模式中，分别用命令 break 和 bafter 在模块的前端和后端设置断点，用 clear 命令来删除断点。

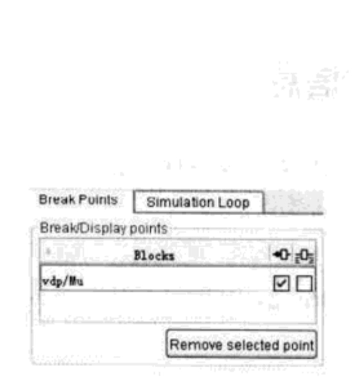


图 5.6 Break Points 面板中设置断点

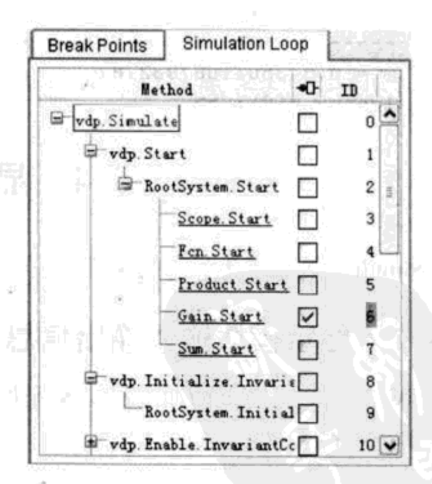


图 5.7 Simulation Loop 面板中设置断点

5.3.2 条件中断

条件中断与非条件中断的不同在于：条件中断中，模型是否发生中断还不确定，取决

于运行时的状况，每次发生中断的情况也不相同，但这些不会改变断点。例如，用户使用 break 命令设置好的断点，不管在什么情况下，当模型运行到断点时都会自动暂停。用于条件中断的命令有 nanbreak、xbreak 和 zcbreak。

1. nanbreak

当模型仿真过程中出现无穷大时，用 nanbreak 命令来设置断点。当设置好断点以后，当仿真出现溢出时，调试器会自动中断仿真。

2. xbreak

使用 xbreak 命令设置的中断主要是针对变步长求解器，当求解器所采用的步长超过模型的步长限制时，仿真就会自动中断。

3. zcbreak

当需要对模型的过零情况进行中断时，可使用 zcbreak 命令。暂停之后，会在命令窗口显示中断在模型中的位置，时间以及过零点是上升还是下降。

例如，在显示模型 zeroxing.mdl 中设置一个过零点断点。

```
>> sldebug zeroxing
%----------------------------------------------------------------%
[TM = 0                    ] zeroxing.Simulate
(sldebug @0): >> zcbreak
Break at zero crossing events            : enabled
(sldebug @0): >> c
ZeroCrossing Events detected. Interrupting model execution
%----------------------------------------------------------------%
[Tm = 0.4                  ] zeroxing.zc.SearchLoop
(sldebug @68): >> c
%----------------------------------------------------------------%
[Tm = 0.3435011087932787   ] zeroxing.zc.SearchLoop
(sldebug @68): >>
```

5.4 显示仿真信息

Simulink 调试器为显示模块状态、模块的输入输出以及其他信息提供了一些命令。

5.4.1 显示模块输入输出信息

调试器可以很方便的通过快捷按钮来显示输入输出信息，如图 5.8 所示。或者通过命令显示，如表 5.3 所示。



图 5.8 显示输入和输出信息按钮

表 5.3 信息显示命令

命　令	功　能
probe	显示被选择模块的当前输入和输出
disp	显示每一个断点的输入输出
trace	任何时候执行模块

1. 显示被选择模块的输入输出信息

为了显示被选择模块的输入输出信息，在 Simulink 调试器模式时可单击快捷按钮，在 MATLAB 命令窗口模式时，可以使用 probe 命令，probe 命令的具体用法如表 5.4 所示。

表 5.4 probe 命令的具体用法

命　令	用　法
probe	进入或者退出 probe 模式。当处于 probe 模式时，调试器可以显示任何被选择模块的当前信息。此时在提示符后面输入任何命令都会使调试器退出 probe 模式。
probe gcb	显示被选中模块的输入输出
probe s:c	显示索引 s:c 指向模块的输入输出

调试器将被选择模块的输入输出及状态信息显示在调试器的 Outputs 面板(Simulink 模式)或者 MATLAB 命令行中(MATLAB 命令模式)。在需要查看那些没有显示信息的模块的信息时，probe 命令将会非常有用。例如，用 step 命令运行模型中的一个接一个的方法时，每次只能看到当前模块的输入与输出，而 probe 命令可以查看其他模块的输入输出信息。

2. 在断点处显示模块的输入输出信息

在任何仿真被中断的时候，disp 命令可以让调试器显示指定模块的输入输出。用户可以通过输入模块的索引，或者在模型窗口选择模块，或者输入带参数 gcb 的 disp 命令。用户可以使用 undisp 命令来移除任何调试器列表中的模块。例如，要删除模块 0:0，或者直接在模型窗口中选择并输入 undisp gcb 命令，或者简单地输入 0:0。

3. 观察模块的输入输出

为了在仿真过程中不中断地观察模块信息，可以选择需要观察的模块，然后单击【调试器】快捷按钮，或者直接在 MATLAB 命令窗口中输入 trace。在 Simulink 窗口调试模式中。如果模块存在断点，可以选中 Break/Display points 面板中相应模块的复选框。还可以直接在 trace 命令中指定模块的索引。相反，如果要删除 trace points 列表下某个模块的显示，可以用 untrace 命令。

5.4.2 显示代数环信息

atrace 命令可以使调试器在每一个时间步内显示模型中出现代数环的求解信息。atrace 命令只有一个参数，用法如表 5.5 所示。

表 5.5　atrace 命令的显示信息

命　令	显示信息
atrace 0	不显示信息
atrace 1	环内变量的解，求解环所需的迭代次数以及估计的解的误差
atrace 2	环内变量的解，求解环所需的迭代次数以及估计的解的误差
atrace 3	在 atrace 2 所显示的信息的基础上加上环路的 Jacobi 矩阵
atrace 4	在 atrace 2 所显示的信息的基础上，加上环路变量在每次迭代的瞬时解

5.4.3　显示系统状态

states 调试命令在 MATLAB 命令窗口列出状态的当前值，如图 5.9 所示。例如，下面的命令是用来显示 Simulink 的反弹球实例模型(bounce)在第一和第二个仿真时间步的状态。

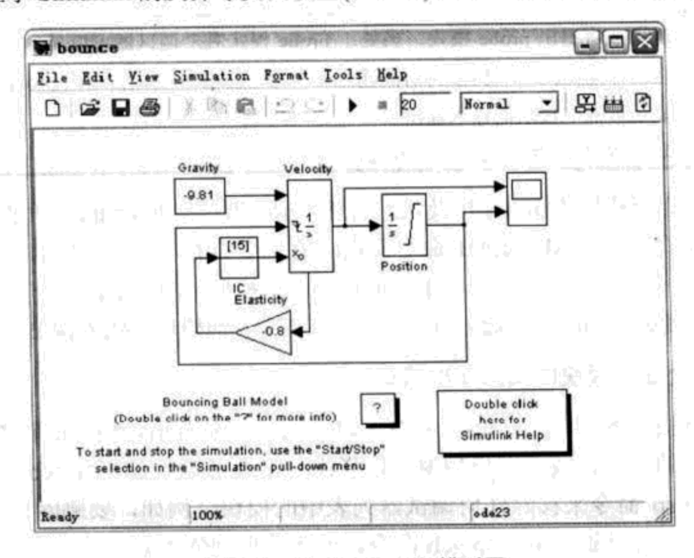


图 5.9　实例 bounce 的模型图

```
>> sldebug bounce
%----------------------------------------------------------------%
[TM = 0                   ] bounce.Simulate
(sldebug @0): >> states
Continuous States:
Idx  Value                 (system:block:element  Name   'BlockName')
  0  0                    (0:0:0  CSTATE  'bounce/Position')
  1  0                    (0:3:0  CSTATE  'bounce/Velocity')
(sldebug @0): >> c
%----------------------------------------------------------------%
[TM = 20                  ] bounce.Simulate
(sldebug @0): >> states
Continuous States:
Idx  Value                 (system:block:element  Name   'BlockName')
  0  0.005296808914250595    (0:0:0  CSTATE  'bounce/Position')
  1  0.1810078006862172      (0:3:0  CSTATE  'bounce/Velocity')
(sldebug @0): >> stop
%----------------------------------------------------------------%
```

```
% Simulation stopped
```

5.4.4　显示积分信息

ishow 命令用来显示积分信息。当被激活时，这个选项就会显示每一时间步或者在遇到约束仿真时间步大小的状态时的时间信息。对于前者，调试器显示仿真时间步的大小。

5.5　显示模型信息

除了能够显示仿真信息之外，调试器还能够提供模型的信息，例如显示模型中模块的执行次序表，显示模块，显示模型中的非虚拟系统和非虚拟模块，显示可能的具有过零点的模块，还显示代数环以及显示调试器的设置。

5.5.1　显示模型中模块的执行顺序

在 Simulink 模型调试器中，Sorted List 面板显示了模型中的模块和每一个虚拟子系统。每一个表列举了子系统所包含的模块，这些模块是按照计算顺序，字母顺序或者其他模块排列方式。在 MATLAB 命令调试模式，可以利用 slist 命令来显示模型的排列列表。

例如显示 bounce 模型中模块的排列列表：

```
>> sldebug bounce
%----------------------------------------------------------------%
[TM = 0                      ] bounce.Simulate
(sldebug @0): >> slist
---- Sorted list for 'bounce' [6 nonvirtual blocks, directFeed=0]
  0:0    'bounce/Position' (Integrator, tid=0)
  0:1    'bounce/Elasticity' (Gain, tid=0)
  0:2    'bounce/IC' (InitialCondition, tid=0)
  0:3    'bounce/Velocity' (Integrator, tid=0)
  0:4    'bounce/Scope' (Scope, tid=0)
  0:5    'bounce/Gravity' (Constant, tid=1)
```

5.5.2　显示模型中的非虚拟系统

systems 命令可以显示模型中正在被调试的非虚拟系统列表。例如，clutch 演示模型包含了下列系统，clutch 模型图如图 5.10 所示，MATLAB 命令操作方式如下：

```
>> sldebug clutch
%----------------------------------------------------------------%
[TM = 0                      ] clutch.Simulate
(sldebug @0): >> systems
 0   'clutch'
 1   'clutch/Locked'
 2   'clutch/Unlocked'
(sldebug @0): >>
```

说明： 对于已经封装后的子系统，Simulink 就会将它们作为模块来处理，同样不作为非虚拟系统来显示。

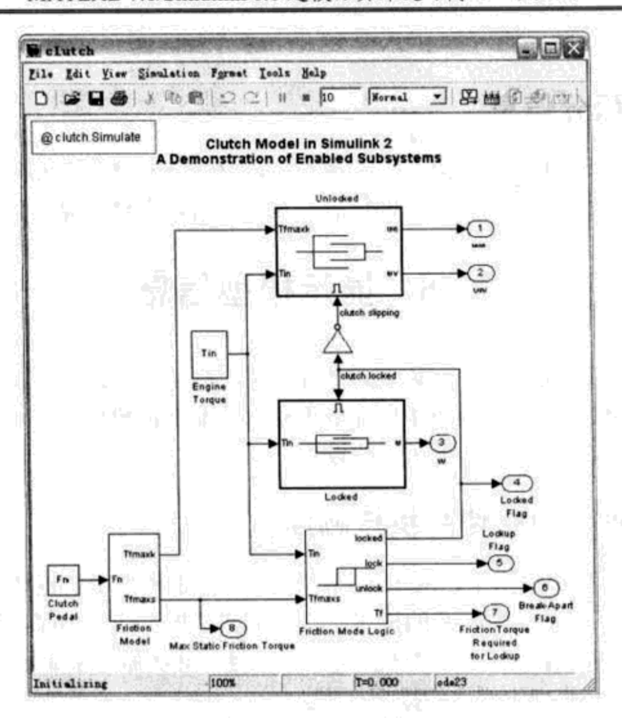


图 5.10 clutch 模型图

5.5.3 显示具有过零点的潜在模块

在仿真过程中，zclist 命令可以列出模型中所有可能会产生非采样过零点的模块的列表。例如，zclist 命令可以列出 clutch 模型的所有产生过零点的潜在模块。

```
>> sldebug clutch
%----------------------------------------------------------------%
[TM = 0                    ] clutch.Simulate
(sldebug @0): >> zclist
 0  0:4:0  R,F  HitCross  'clutch/Friction Mode Logic/Lockup
Detection/Velocities Match'
 1  0:4:1    R
 2  0:10:0  R,F  Abs  'clutch/Friction Mode Logic/Lockup
Detection/Required Friction for Lockup/Abs'
 3  0:12:0  R,F  RelationalOperator  'clutch/Friction Mode Logic/Lockup
Detection/Required Friction for Lockup/Relational Operator'
 4  0:19:0  R,F  Abs  'clutch/Friction Mode Logic/Break Apart
Detection/Abs'
 5  0:20:0  R,F  RelationalOperator  'clutch/Friction Mode Logic/Break
Apart Detection/Relational Operator'
 6  2:3:0  R,F  Signum  'clutch/Unlocked/slip direction'
(sldebug @0): >>
```

5.5.4 显示代数环

ashow 命令能够高亮显示一个特定的代数环或者包括某个特定模块的代数环。高亮显示一个特定代数环的方法是 ashow s#n，其中 s 是指包括次代数环的系统索引，n 指代数环在该系统内的索引。要高亮显示包含当前选定模块的代数环，可以使用 ashow gcb 命令。

如果要高亮显示包含指定模块的代数环时，可以使用 ashow s:b 命令，其中 s:b 是模块的索引。如果要消除某个模块的高亮显示，可以使用 ashow clear 命令。

5.5.5 显示调试器状态

在 Simulink 交互模式中，调试器在 Status 面板中显示了不同调试选项，如条件断点。在 MATLAB 命令模式中，可以使用 status 命令来显示调试器的设置，例如下面的命令显示了 vdp.mdl 模型的初始设置。

```
>> sim('vdp',[0,10],simset('debug','on'))
%----------------------------------------------------------------%
[TM = 0                    ] vdp.Simulate
(sldebug @0): >>  status
%----------------------------------------------------------------%
Current simulation time                     : 0 (MajorTimeStep)
Default command to execute on return/enter : ""
Break at zero crossing events                : disabled
Break on failed integration step             : disabled
Time break point                           : disabled
Break on non-finite (NaN,Inf) values          : disabled
Break on solver reset request                : disabled
Display level for disp, trace, probe          : 1 (i/o, states)
Solver trace level                          : 0
Algebraic loop tracing level                 : 0
Animation Mode                             : off
Window reuse                               : not supported
Execution Mode                             : Normal
Display level for etrace                     : 0 (disabled)
Break points                               : none installed
Display points                             : none installed
Trace points                               : none installed
```



第 6 章　Simulink 模型仿真

经过前面几章的介绍，下面通过一些具有代表性的实例来介绍如何综合应用前面的知识来建造一个仿真模型。

本章主要内容包括：

- 仿真的基本过程
- 对单自由度系统进行仿真
- 多自由度系统进行仿真
- 利用 Simulink 中的 If 条件模块
- 利用 Simulink 求解微分-代数方程

6.1　仿真的基本过程

建立一个模型应该按照一定的顺序，这样才能够不会遗漏某些步骤，下面给出一个创建 Simulink 模型的基本过程，这个过程并不是惟一的，可根据个人的爱好而定，基本操作步骤如下：

(1) 根据系统具体情况，建立数学仿真模型。

(2) 打开一个空白模型编辑窗口，如图 6.1 所示。

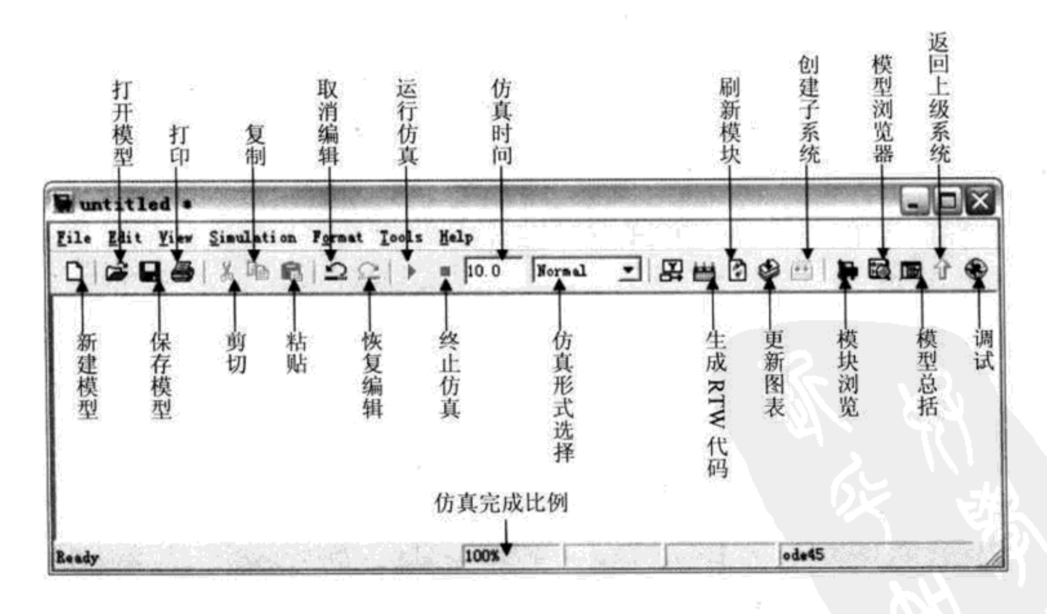


图 6.1　Simulink 工具栏功能说明图

(3) 拖放模块建立模型。

(4) 设置模块参数。
(5) 对模块进行连线。
(6) 设置仿真模型的系统参数。
(7) 运行仿真。
(8) 查看仿真结果。
(9) 保存文件退出。

6.2　对单自由度系统进行仿真

本节对系统仿真采用积分模块，6.3 节仿真采用状态空间模块。对比两个实例，可以发现它们在具体处理过程中的不同。本节将会介绍如何建立一个动力学系统，在此过程中，还会看到如何通过命令流和鼠标来一步一步地建造模型。本例为了综合说明模块的用法，不拘泥于最简单的形式。操作步骤如下：

(1) 建立系统的数学模型。

建立动力学方程如下。

$$m\frac{d^2y}{dt^2}+c\frac{dy}{dt}+ky=F$$

其中 $m=1$ 为质量，$c=4$ 为阻尼，$k=3$ 为刚度，F 为外激励，取 $F=2\sin(2t+\pi/3)$。

将参数代入，并转换形式为：

$$m\frac{d^2y}{dt^2}=F-c\frac{dy}{dt}-ky$$

$$\frac{d^2y}{dt^2}=2\sin(2t+\pi/3)-4\frac{dy}{dt}-3y$$

(2) 打开一个一个空白模型编辑窗口，保存文件名为 model06to01.mdl。

(3) 创建相应模块，如图 6.2 所示。

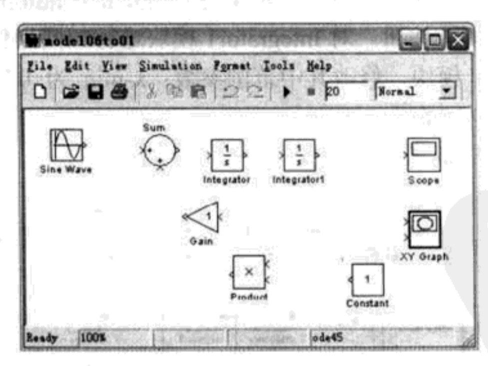


图 6.2　仿真模型模块图

(4) 对模块进行以下设置。

- 设置 Sine Wave 模块，双击模块弹出参数的对话框，设置如图 6.3 所示，参数 Amplitude 为 2，参数 Bias 为 0，参数 Frequency 为 2，参数 Phase 为π/3，然后单击 OK 按钮。

- 对 Sum 模块进行设置，双击模块弹出设置参数的对话框，参数设置如图 6.4 所示，参数 Icon shape 选择 rectangular 选项、List of signs 为“+−−”，然后单击 OK 按钮。

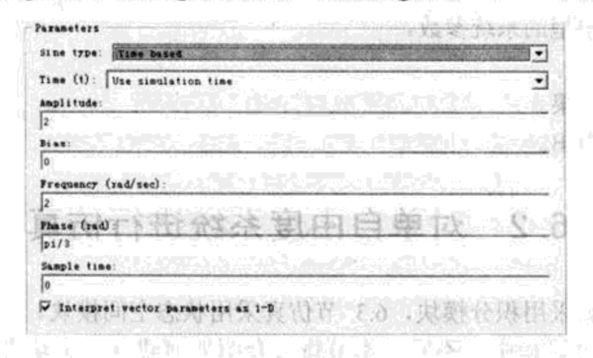


图 6.3　Sine Wave 模块设置对话框

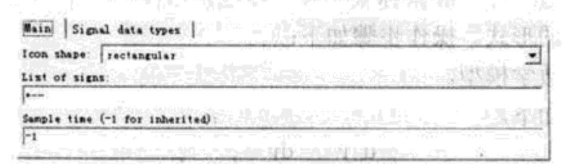


图 6.4　Sum 模块设置对话框

- 设置 Constant 模块，双击模块弹出参数对话框，设置 Constant value 参数为 3，单击 OK 按钮。用右键单击模块，在弹出浮动菜单中单击 Format | Flip Block。
- 设置 Gain 模块，双击模块弹出设置参数的对话框，设置 Gain 为 4，单击 OK 按钮。用右击模块，在弹出的菜单中选择 Format | Flip Block 命令。
- 设置 Product 模块，右击模块，在弹出的菜单中选择 Format | Flip Block 命令。
- 设置 Integrator 和 Integrator1 模块，双击 Integrator 模块，弹出设置参数的对话框。在 Integrator 模块的参数对话框中设置 Initial condition 参数为 0.5，即初始速度为 0.5，单击 OK 按钮。对 Integrator1 模块进行相同操作，即初始位移为 0.5。
- 设置 XY Graph 模块，修改其中范围参数：x-min 为−0.7，x-max 为 0.7，y-min 为−0.5，y-max 为 0.5。

(5) 连接模块，如图 6.5 所示。

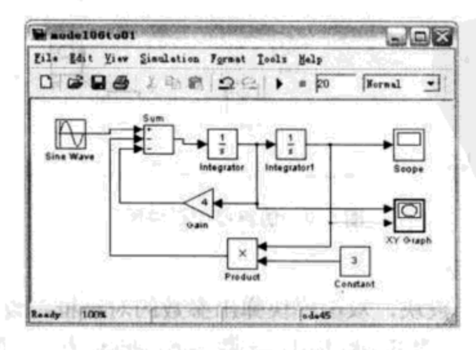


图 6.5　系统仿真模型结构图

(6) 对模型的系统参数进行设置，选择 Simulation | Configuration Parameters 命令，弹出 Configuration Parameters 对话框如图 6.6 所示。

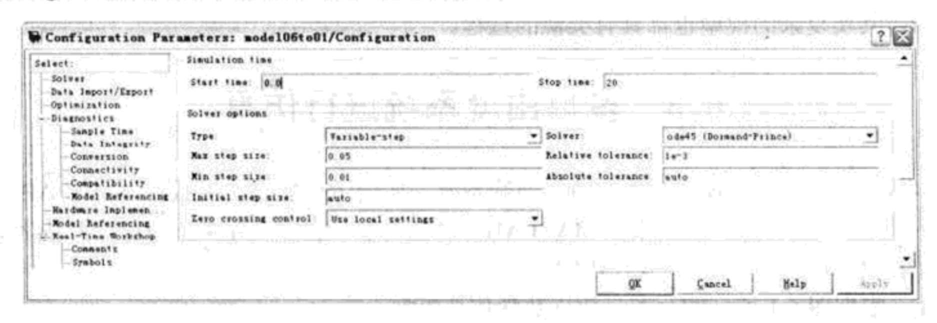


图 6.6 Configuration Parameters 对话框

- Start time 为仿真开始时间，在此取默认值 0。
- Stop time 为仿真结束时间，在修改为 20。
- Type 为是否固定步长，在此接受默认选项 Varaible-step。
- Solver 为计算方法，在此取默认 ode45 方法；各种方法的具体不同，可查看本书相关章节或者 MATLAB 帮助。
- Max step size 和 Min step size 为变步长的最大值和最小值，分别设置为 0.05 和 0.01。
- Relative tolerance 和 Absolute tolerance 分别为相对误差和绝对误差，在此取默认值。
- Initial step size 初始步长大小，在此取默认值。
- Zero crossing control 取默认值。

说明： 以上设置只是图 6.6 中左边菜单中 Solver 参数，还有其他选项，在此都不使用。

(7) 运行仿真。单击 ▶ 按钮，或者选择 Simulation | Start 命令，或者按 Ctrl+T 快捷键。

(8) 查看运行结果。X-Y Graph 会自动弹出运行结果，如图 6.7 所示，横坐标是位移，纵坐标是速度。双击 Scope 模块，会弹出如图 6.8 所示位移时程曲线，单击图中的望远镜按钮，可以自动调整坐标的大小。

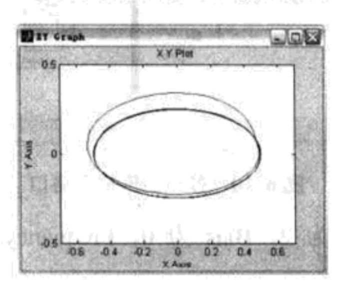


图 6.7 系统相轨迹图

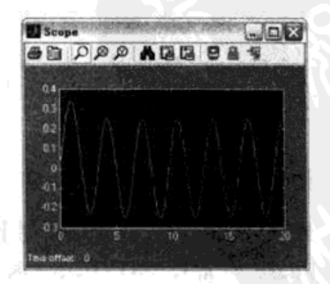


图 6.8 系统时程轨迹图

(9) 保存文件。

至此我们已经完成了一个简单模型的建立、模块设置、模型设置和仿真以及最后的结果显示，其实复杂的模型也都具有类似的过程。

6.3 多自由度系统进行仿真

本节实例采用状态模块来描述质量弹簧系统，如图 6.9 所示，参数从 MATLAB 工作空间中获取，最后结果将返回到 MATLAB 工作空间。这一节所涉及的难度和内容都较 6.2 节的内容有较大的提高。操作步骤如下：

(1) 根据图 6.1 所示的系统，建立数学仿真模型。

其中系统参数为 $m=1$，$c_1=c_2=c_3=4$，$k=3$，先建立动力学方程，然后将动力学方程转化到状态方程当中。

$$m\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\begin{Bmatrix}\ddot{x}_1\\\ddot{x}_2\\\ddot{x}_3\end{Bmatrix}+c\begin{bmatrix}2&-1&0\\-1&2&-1\\0&-1&1\end{bmatrix}\begin{Bmatrix}\dot{x}_1\\\dot{x}_2\\\dot{x}_3\end{Bmatrix}+k\begin{bmatrix}2&-1&0\\-1&2&-1\\0&-1&1\end{bmatrix}\begin{Bmatrix}x_1\\x_2\\x_3\end{Bmatrix}=\begin{Bmatrix}0\\0\\1\end{Bmatrix}u$$

即为：$M\ddot{X}+P\dot{X}+KX=Gu$，

将仿真转换成状态方程：

$$\begin{cases}\dot{Z}(t)=AZ(t)+BU(t)\\Y(t)=CZ(t)+DU(t)\end{cases}$$

其中：$Z(t)=\begin{bmatrix}X(t)\\\dot{X}(t)\end{bmatrix}$，$A=\begin{bmatrix}0&I\\-M^{-1}K&-M^{-1}P\end{bmatrix}$，$B=\begin{bmatrix}0\\M^{-1}G\end{bmatrix}$，$C=I_{6\times6}$，$D=0_{6\times1}$。

(2) 打开一个空白模型编辑窗口。

(3) 将 6.2 节中(3)、(4)和(5)步合并起来，得到如图 6.10 所示模型图。

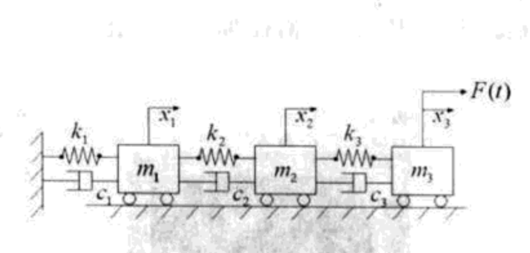


图 6.9 多自由度系统实物图

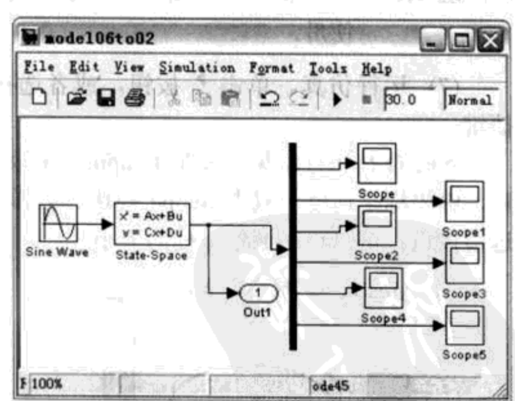


图 6.10 多自由度模型窗口

- 模块 Sine Wave 模块参数设置：Amplitude 为 2，Bias 为 0，Frequency 为 2，Phase 为 π/3，然后单击 OK 按钮。
- 模块 State-Space 模块参数：A 为 A，B 为 B，C 为 C，D 为 D，Initial Conditions

为[0.5;0.5;0.5;0;0;0]，Absolute tolerance 为 auto，如图 6.11 所示。

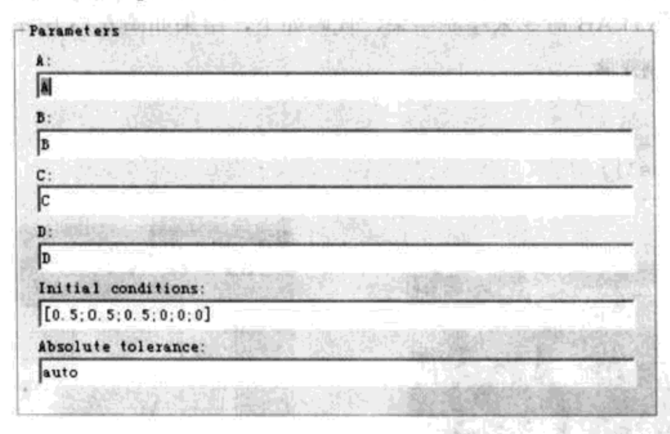


图 6.11　模块 State-Space 模块参数设置

其中参数 A、B、C、D 均来自 MATLAB 工作空间。在 MATLAB 命令空间中输入如下代码：

```
m=1;
c=4;
k=3;
M=m*eye(3);
P=c*[2 -1 0;-1 2 -1;0 -1 1];
K=k*[2 -1 0;-1 2 -1;0 -1 1];
G=[0;0;1];
A=cat(1,cat(2,zeros(3,3),eye(3)),cat(2,-inv(M)*K,-inv(M)*P));
B=cat(1,zeros(3,1),-inv(M)*G);
C=eye(6);
D=zeros(6,1);
```

(4) 设置仿真模型的系统参数。

- Start time 为仿真开始时间，在此取默认值为 0。
- Stop time 为仿真结束时间，修改为 30。
- Type 为是否固定步长，在此取默认 Varaible-step。
- Solver 为计算方法，在此取默认 ode45 方法；各种方法的具体不同，可查看 MATLAB 帮助。
- Max step size 和 Min step size 为变步长的最大值和最小值，在此分别设置为 0.05 和 0.01。
- Relative tolerance 和 Absolute tolerance 分别为相对误差和绝对误差，在此取默认值。
- Initial step size 为初始步长大小，在此取默认值。
- Zero crossing control 取默认值。

(5) 运行仿真。

(6) 查看仿真结果。双击相应的 Scope 模块，可以查看相应的状态变量，如图 6.12 所示。

由于在模型文件中使用了 Out1 模块，输出结果已经保存到 MATLAB 工作空间当

中，时间变量输出为 tout，状态变量为 yout。

可以使用 MATLAB 命令来绘制结果，代码如下，结果如图 6.13 所示。

```
k=1;%选择绘制变量
plot(tout,yout(:,k))
grid on
axis([0 30 -0.4 0.6]);
xlabel('Time');
ylabel('State');
```

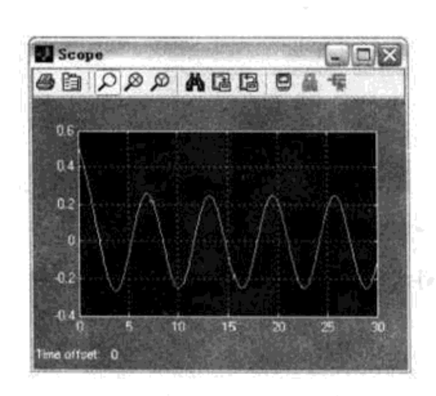


图 6.12 Simulink 窗口 Scope 显示状态 1 仿真结果图

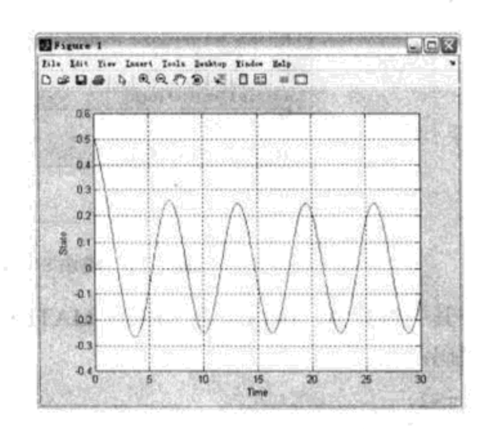


图 6.13 MATLAB 命令窗口绘制状态 1 仿真结果图

比较图 6.12 和图 6.13 可以看出，结果一模一样。

(7) 保存文件。

至此已经完成了一个较为复杂模型的建立、模块设置、模型设置和仿真，以及最后的结果显示。用户知道如何从 MATLAB 工作空间中调入数据，以及如何将仿真结果输出到 MATLAB 工作空间中。

6.4 利用 Simulink 中的 If 条件模块

该实例演示了一个正弦信号是如何输入到 If 模块的。在运行仿真之后，Scope 模块就会出现 3 个结果图。这个演示主要是为了说明带有使能子系统(Enabled subsystems)的 If 模块是如何装配和设置的。通过这个实例中，举一反三就能够了解其他诸如触发子系统等的使用方法。

在 3 个结果图中，第一个结果图绘制的是原始正弦信号结果图(假定为曲线 A)，以及在原始信号绝对值基础上偏移 2 的结果图(曲线 B)。但是在 0~2.5s 和 5~7.5s 时，曲线 B 中的正弦信号只在子系统执行的时候才会求绝对值，当子系统没有被激发的时候，正弦信号则为 0。第 2 个结果图则是激发信号图。当脉冲激发信号为 1 时，If 子系统被激发，当为 0 时，Else 子系统被激发。第 3 个结果图绘制的是原始正弦信号结果图(假定为曲线 C)，以及在原始信号取饱和基础上偏移 2 的结果图(曲线 D)。曲线 D 只在 2.5~5s 和 7.5～10s 时，Else 子系统被激发时才执行。

将正弦信号输入到 If 模块的操作步骤如下：

(1) 搭建仿真模型图，如图 6.14 所示，保存文件为 model06to03.mdl。

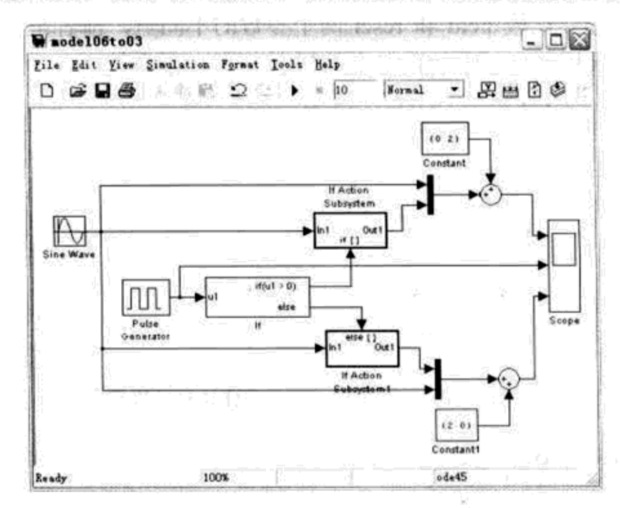


图 6.14　仿真模型结构图

注意，Scope 模块通常只有一个输入端，这里需要修改其参数。双击 Scope 模块，在弹出 Scope 窗口中单击按钮，则会弹出如图 6.15 所示对话框，将参数 Number of axes 设置为 3，然后单击 OK 按钮。



图 6.15　Scope Parameters 对话框

(2) 设定参数。

- 双击 Pulse Generator 模块，在弹出的对话框中设置参数，如图 6.16 所示，Amplitude 为 1，Period 为 5，Pulse Width 为 50，Phase delay 为 0，然后单击 OK 按钮。
- 双击 Constant 模块，在弹出的对话框中设定参数 Constant Vaule：[0 2]，然后单击 OK 按钮。
- 双击 Constant1 模块，在弹出的对话框中设定参数 Constant Vaule：[2 0]，然后单击 OK 按钮。
- 双击 If Action Subsystem 模块，在弹出子系统模型窗口中添加绝对值(Abs)模块，如图 6.17 所示。双击子系统中的 Abs 模块，对话框设定如图 6.18 所示，两个选

项全部选定，然后单击 OK 按钮。双击子系统中的 Action Port 模块，在弹出对话框中设定参数 States when execution is resumed 为 reset，如图 6.19 所示，然后单击 OK 按钮。比较 held 和 reset 两个参数的不同点，前者是保持当前值，后者则是回零。最后单击子系统窗口中的🖫按钮保存子系统。

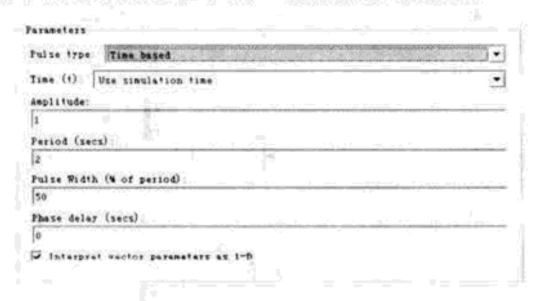


图 6.16　Pulse Generator 模块参数设置

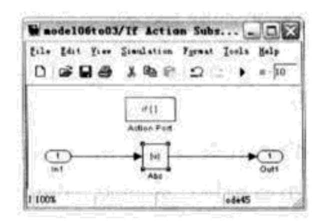


图 6.17　If Action Subsystem 子系统

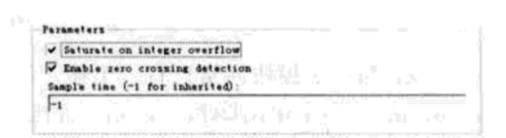


图 6.18　Abs 模块参数设置



图 6.19　Action Port 模块参数设置

- 双击 If Action Subsystem1 模块，在弹出子系统模型窗口中添加饱和(Saturation)模块，如图 6.20 所示。双击子系统中的 Saturation 模块，对话框设置如图 6.21 所示，设置 Upper limit 为 0.75，Lower limit 为-0.75，然后单击 OK 按钮。然后双击子系统中的 Action 模块，在弹出对话框中设置参数 States when execution is resumed 为 held，如图 6.22 所示，然后单击 OK 按钮。最后单击子系统窗口中的🖫按钮保存子系统。

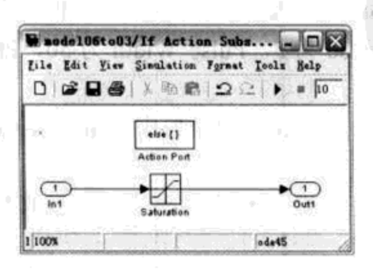


图 6.20　If Action Subsystem1 子系统

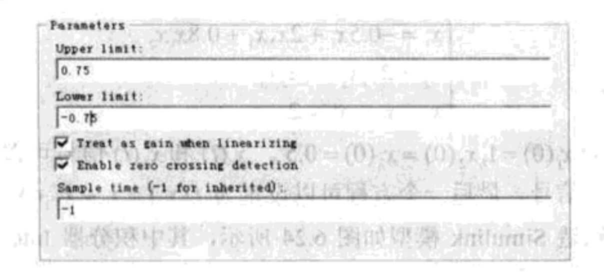


图 6.21　Saturation 模块参数设定

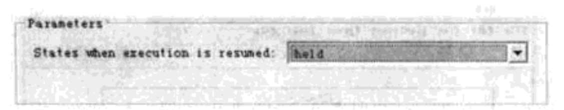


图 6.22　Action Port 模块参数设定

(3) 运行仿真，结果如图 6.23 所示。

(4) 保存文件。

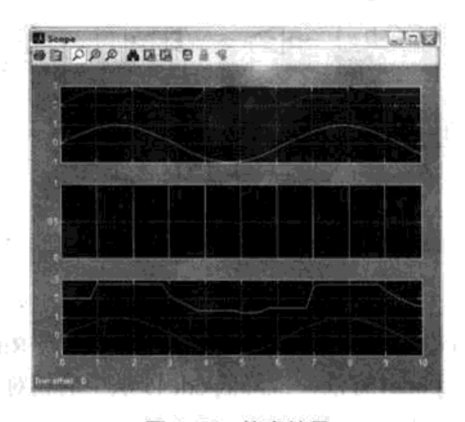


图 6.23　仿真结果

至此已经完成了一个使能系统模型的建立、模块设置、模型设置和仿真，以及最后的结果显示，至于其他类似触发模型，以及混合模型的仿真，都可以通过此例进行举一反三，触类旁通。

6.5　利用 Simulink 求解微分-代数方程

6.2 节和 6.3 节介绍的实例都是纯微分方程，不含代数方程，通常情况下 Simulink 模型中应该避免代数方程的出现，以免出现代数环。然而微分代数混合方程问题在机器人等领域又是经常遇到，为了直接利用 Simulink 解决微分代数混合方程问题，可以利用代数约束(Algebraic Constrain)模块。

现有如下微分－代数混合方程：

$$\begin{cases} \dot{x}_1 = -0.5x_1 + 2x_2x_3 + 0.8x_1x_2 \\ \dot{x}_2 = 2x_1x_2 - 5x_2x_3 - x_2^2 \\ x_1 + x_2 + x_3 = 2 \end{cases}$$

已知初始条件为 $x_1(0)=1, x_2(0)=x_3(0)=0.5$。$x_1(t)$ 和 $x_2(t)$ 信号可以设置为积分器的输出，可将其视为已知信号。最后一个方程可以设定为 $f(x_3)=x_1+x_2+x_3-2=0$。

根据以上方程构造 Simulink 模型如图 6.24 所示，其中积分器 Integrator 和 Integrator1 初始值分别设置为 1 和 0.5。

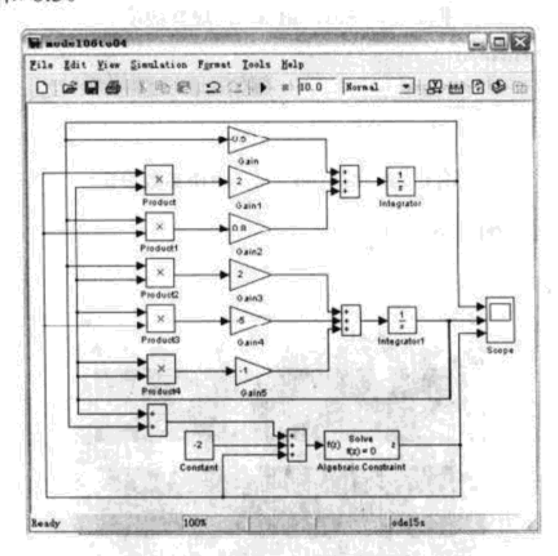


图 6.24 系统仿真模型图

此类微分-代数混合方程直接采用默认的 ODE45(四阶五级 Runge-Kutta 法)得出的结果不是很可靠，在仿真时采用 ode15s 法，或降低 ode45 法中的相对误差要求。

运行仿真结果如图 6.25 所示。

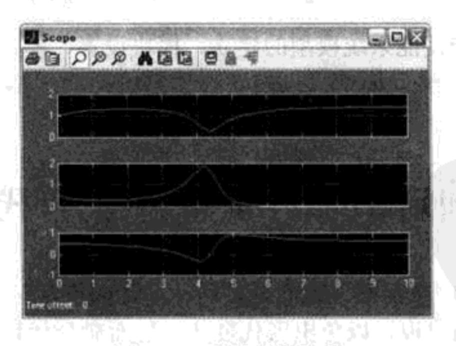


图 6.25 系统仿真结果

至此已经完成了一个微分-代数混合方程模型的建立、模块设置、模型设置和仿真以及最后的结果显示。这也为我们提供了一种解决代数环问题的方法，至于代数环问题，后面有专门的章节进行讲解。

第7章　Simulink子系统封装技术

对于简单的系统来说，可以直接建立系统的模型，并分析模块之间的相互关系以及模块的输入输出关系。但是对一个复杂系统来说，或者一个大系统中存在多个相对独立的子系统时，Simulink 中将会包含非常多的模块，使得各个模块之间的相互关系显得非常复杂，不利于分析。而子系统则正是针对以上原因而设计的，可以将联系比较紧密的模块、或者归属于一个子系统的模块进行封装，这样就能够对大系统模型一目了然。

本章主要内容包括：

- Simulink 子系统简介
- Simulink 高级子系统应用
- Simulink 精装子系统
- 精装子系统实例
- Simulink 模块库技术

7.1　Simulink子系统简介

7.1.1　建立子系统

Simulink 提供的子系统功能可以大大地增强 Simulink 系统模型框图的可读性，可以不必了解子系统中每个模块的功能就能够了解整个系统的框架。

子系统可以理解为一种“容器”，我们可以将一组相关的模块封装到这个子系统模块当中，并且等效于原系统模块群的功能，而对其中的模块我们可以暂时不去了解。组合后的子系统可以进行类似模块的设置，在模型仿真过程中可作为一个模块。

下面来介绍两种建立子系统的方法。

1. 在已有的系统模型中建立子系统

设已有 Simulink 模型图如图 7.1 所示，选择框选需要封装的模块区域(用 Shift 键和鼠标左键配合可以达到同样的目的)，框选如图 7.2 所示区域。然后右击，弹出浮动菜单，选择 Creat Subsystem 命令，如图 7.3 所示。创建子系统的结果如图 7.4 所示。然后双击 Subsystem 模块，弹出子模型如图 7.5 所示。

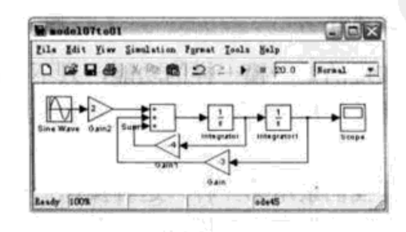


图7.1　简单Simulink模型

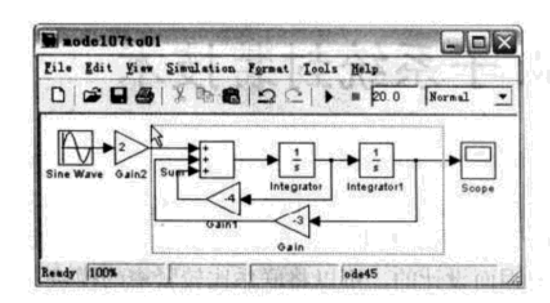


图 7.2 框选需要封装的模块



图 7.3 快捷菜单

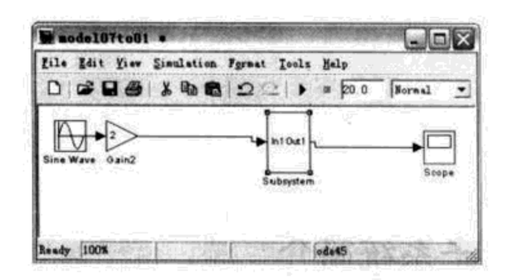


图 7.4 创建子系统后的模型

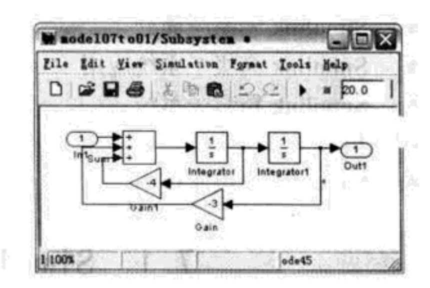


图 7.5 子系统模型图

很明显，子系统的功能就是把相关模块集中起来，并没有删除。这个系统的结构仍然没有变。

2. 在已有的系统模型中新建子系统

简单建立模型如图 7.6 所示。双击 Subsystem 模块，然后在弹出的窗口中建立如图 7.7 所示模型。

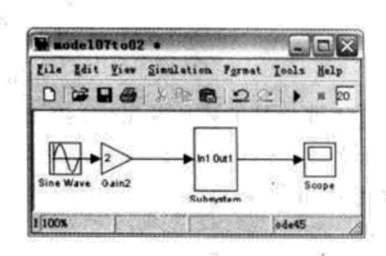


图 7.6 含有子系统的模型

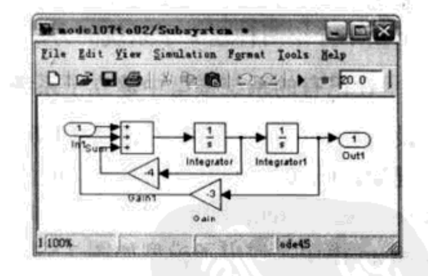


图 7.7 子系统模型结构

这两种创建子系统最后实现的是一模一样的功能，只不过操作顺序不同。前者是先将这个结构搭建起来，然后将相关的模块封装起来；后者则是先做一个封装容器，然后在封装容器中添加模块。

对于一个相对简单的模型我们采用第一种，这种操作一般不容易出错，能够顺利搭建模型。而对于非常复杂的系统，我们事先将模型分成若干个子系统，然后再采用第二种方法进行建模。

7.1.2　子系统的基本操作

在使用 Simulink 子系统建立系统模型时，有如下几个常用的操作：

- 子系统命名：命名方法与模块命名类似，是拥有代表意义的文字来对子系统进行命名，有利于增强模块的可读性。
- 子系统编辑：用鼠标双击子系统模块图标，打开子系统并对其进行编辑。
- 子系统的输入：使用 Sources 模块库中的 Inport 输入模块，即 In1 模块，作为子系统的输入端口。
- 子系统的输出：使用 Sinks 模块库中的 Outport 输出模块，即 Out1 模块，作为子系统的输出端口。

7.2　Simulink 高级子系统应用

子系统的最基本目的就是将一组相关的模块包含到一个模块中，用以简化系统，使得系统的分析更加容易。例如在一个控制系统中，受控系统就可以视为一个子系统，控制器也可以作为一个子系统。从前面的介绍可以发现，这些子系统就如其他一般模块一样，都是具有特定输入输出的模块。对于子系统输入的信号，会产生一个特定的输出信号。但是对于某些特殊的情况，并不是对所有的输入信号都要产生输出信号，只有在某些特定的条件下才会产生输出信号，这就需要输入一个控制信号。控制信号由子系统模块的特定端口输入，这样的子系统称为条件执行子系统。在条件子系统中，输入信号取决于输入信号和控制信号。

根据不同的控制信号，可将条件执行子系统分为如下几类：

- 触发子系统：当控制信号符号发生变化时，执行子系统。具体有以下形式：
  - 控制信号上升沿触发。
  - 控制信号下降沿触发。
  - 控制信号的双边沿触发。
- 使能子系统：当控制信号为正时，子系统开始执行子系统。
- 函数调用子系统：控制信号是由自定义的 S 函数发出调用信号，开始执行信号。

7.2.1　触发子系统

触发子系统就是在控制信号符号发生变化时，执行子系统。下面分别介绍其中的 3 个子系统：

- 控制信号上升沿触发：子系统在控制信号上升的时候执行。
- 控制信号下降沿触发；子系统在控制信号下降的时候执行。
- 控制信号的双边沿触发：子系统在控制信号符号发生变化就执行，不管是上升还是下降。

【例 0701】触发子系统建模实例演示

在本例中同时使用了上述的 3 个子系统(控制信号上升沿触发、控制信号下降沿触发、控

制信号的双边沿触发)，从中可以发现这 3 个子系统之间的区别，模型如图 7.8 所示：

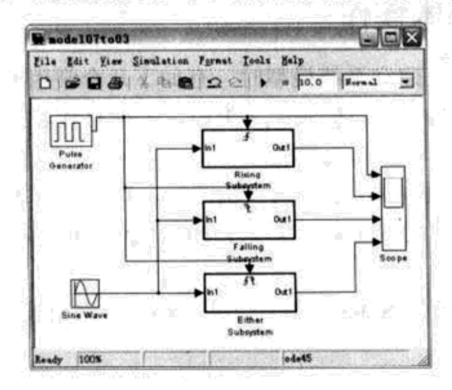


图 7.8　函数触发子系统的 Simulink 模型

1. 进行模块参数设置

(1) Pulse Generator 模块参数设置。其中 Amplitude 为 1，Period 为 1，Pulse Width 为 50，Phase delay 为 0，如图 7.9 所示。

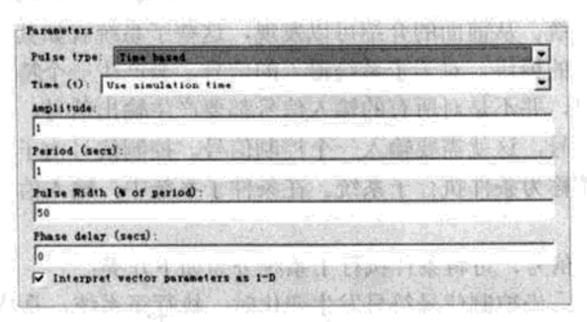


图 7.9　Pulse Generator 模块参数设置

(2) Sine Wave 模块参数设置。其中 Amplitude 为 1，Bias 为 0，Frequency 为 1，Phase Delay 为 0，Sample time 为 0，如图 7.10 所示。

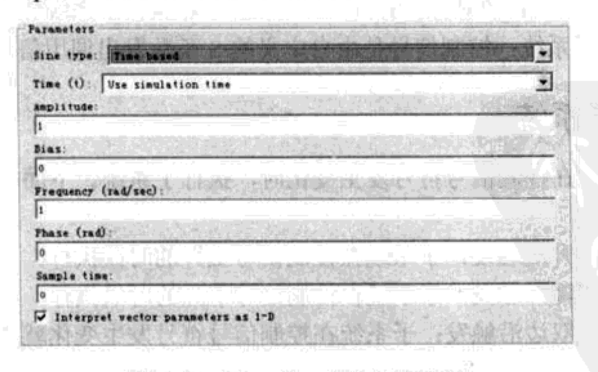


图 7.10　Sine Wave 模块参数设置

(3) Scope 模块设置。双击此模块，在弹出窗口单击按钮，在弹出参数对话框中设

置 Number of axes 为 4。

(4) Rising Subsystem 模块参数设置。双击此模块，弹出窗口如图 7.11 所示，然后双击其中的 Trigger 模块，在弹出的参数对话框中设置 Trigger Type 栏中选项为 rising，然后单击 OK 按钮。

(5) Falling Subsystem 模块参数设置。双击此模块，弹出窗口如图 7.12 所示，然后双击其中的 Trigger 模块，在弹出的参数对话框中设置 Trigger Type 栏中选项为 falling，单击 OK 按钮。

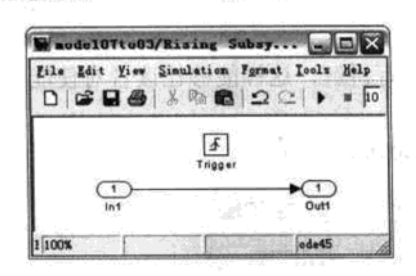



图 7.11　Rising Subsystem 子系统模型

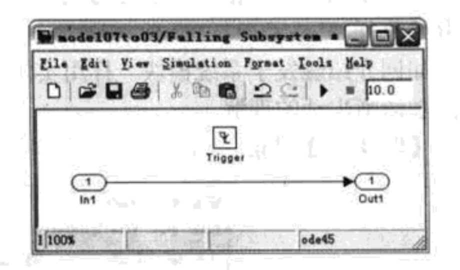


图 7.12　Falling Subsystem 子系统模型

(6) Either Subsystem 模块参数设置。双击此模块，弹出窗口如图 7.13 所示，然后双击其中的 Trigger 模块，在弹出的参数对话框中设置 Trigger Type 栏中选项为 either，然后单击 OK 按钮。

2. 运行仿真

结果如图 7.14 所示。

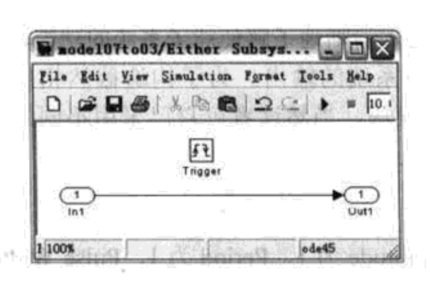


图 7.13　Either Subsystem 子系统模型

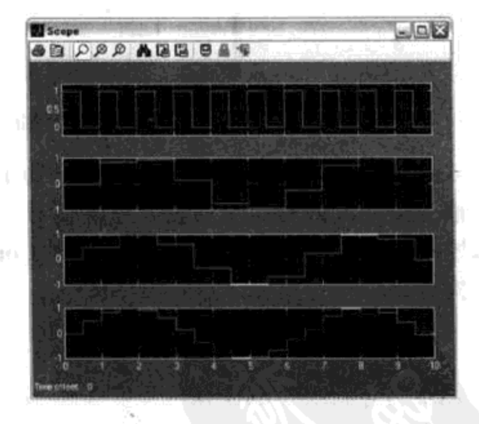


图 7.14　仿真结果

3. 结果分析

从结果图中，第 1 幅图是触发信号，第 2 幅是上升沿触发子系统运行结果，第 3 幅是下降沿触发子系统运行结果，第 4 幅是双边沿触发子系统运行结果。

从中可以看出，在触发信号由 1→0 时，下降沿触发子系统和双边沿触发子系统被执行，当 0→1 时，上升沿触发子系统和双边沿触发子系统被执行。

可以发现：

- 上升沿触发子系统在控制信号 0→1 的时候执行。
- 下降沿触发子系统在控制信号 1→0 的时候执行。
- 双边沿触发子系统在控制信号符号发生变化就执行，不管是 0→1，还是 1→0。

7.2.2 使能子系统

使能子系统，是指输入信号为真时，子系统开始执行，也即输入信号为正数时，子系统开始执行。

如同介绍触发子系统模块一样用实例的形式进行讲解，这样能够让读者对使能子系统有更加清晰生动的理解。

【例 0702】使能子系统建模的文件名为 model07to04.mdl，如图 7.15 所示。

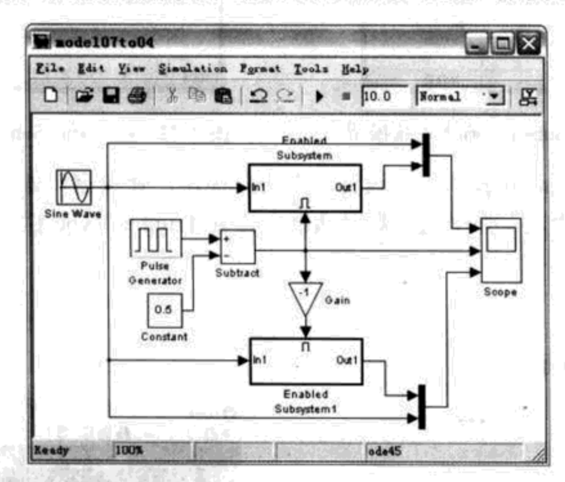


图 7.15 带有使能子系统的 Simulink 模型

在本例中使用了两个使能子系统，为了能够更加清晰地了解使能子系统的功能，对同一输入信号取截然相反的输入控制信号，对比子系统的输出。为了构造截然相反的输入控制信号，我们采用了 Gain 模块和 Constant 模块，当然读者可自己采用不同的模块来构造。

1. 进行模块参数设置

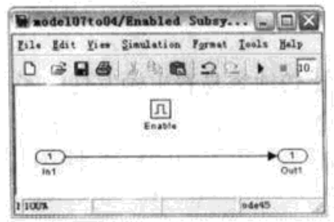
(1) Pulse Generator 模块参数设置。其中 Amplitude 为 1，Period 为 1，Pulse Width 为 50，Phase delay 为 0。

(2) Sine Wave 模块参数设置。其中 Amplitude 为 1，Bias 为 0，Frequency 为 1，Phase delay 为 0，Sample time 为 0。

(3) Constant 模块参数。其中 Constant value 为 0.5；

(4) Gain 模块参数。其中 Gain 为-1。

(5) Enabled Subsystem 模块和 Enabled Subsystem1 模块采用同样的设置。双击则弹出如图 7.16 所示窗口，然后双击 Enable 模块，弹出如图 7.17 所示对话框。



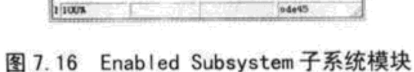


图 7.16　Enabled Subsystem 子系统模块

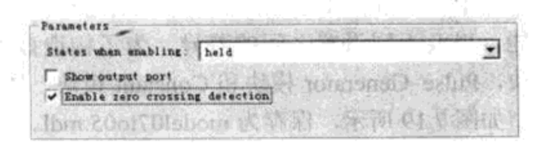


图 7.17　Enable 参数对话框

2. 运行仿真

模型系统参数采用默认值，仿真结果如图 7.18 所示。

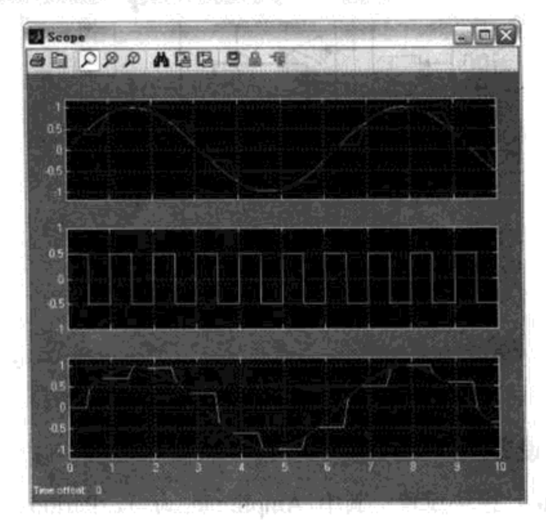


图 7.18　仿真结果

3. 结果分析

从图 7.18 所示的结果图中可见，第 1 幅图为 Pulse Generator 模块和 Constant 求和信号直接输入使能模块的结果图和正弦信号图，第 2 幅图为 Pulse Generator 模块信号，第 3 幅图为 Pulse Generator 模块和 Constant 求和信号取反后输入使能模块的结果图和正弦信号图。

从结果图中可以看出：当 Pulse Generator 模块产生的信号为正时，第 1 个使能子系统直接输出正弦信号，而第 2 个使能子系统的信号则保持不变。当 Pulse Generator 模块产生的信号为负时，情况则正好相反，第 2 个使能子系统直接输出正弦信号，而第 1 个使能子系统的信号则保持不变。

7.2.3 触发使能子系统

所谓触发使能子系统，就是同时融合了触发子系统和使能子系统的功能。

如同介绍触发子系统模块一样用实例的形式进行讲解，这样能够让读者对触发使能子

系统有更加清晰生动的理解。

【例 0703】触发使能子系统建模实例演示。

本例采用的 4 个触发使能子系统进行组合，可以比较地容易看出触发使能子系统的功能，以及不同设置之间的差异。为了构造截然相反的输入控制信号，我们采用 Gain 模块，Pulse Generator 模块和 Constant 模块，同样，用户可采用不同的模块来构造。具体模型如图 7.19 所示，保存为 model07to05.mdl。

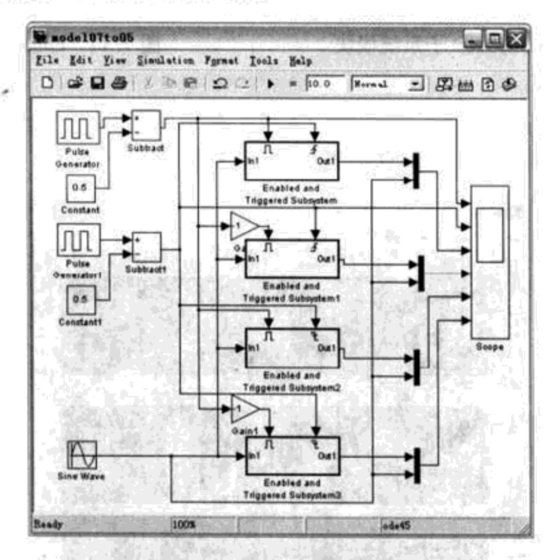


图 7.19 触发使能子系统建模实例演示模型

1. 进行模块参数设置

(1) Pulse Generator 模块设置。其中 Amplitude 为 1，Period 为 1，Pulse Width 为 50，Phase delay 为 0。

(2) Pulse Generator 模块设置。其中 Amplitude 为 1，Period 为 0.5，Pulse Width 为 50，Phase delay 为 0。

(3) Sine Wave 模块设置。其中 Amplitude 为 1，Bias 为 0，Frequency 为 1，Phase delay 为 0，Sample time 为 0。

(4) Constant 模块和 Constant1 模块参数。其中 Constant value 为 0.5。

(5) Gain1 模块参数。其中 Gain 为-1。

(6) Enabled and Triggered Subsystem 模块和 Enabled and Triggered Subsystem1 模块采用同样的设置。双击此模块，则弹出如图 7.20 所示窗口。接着双击 Triggered 模块，弹出如图 7.21 所示对话框，在 Triggered Type 下拉列表框中选择 rising 选项。然后双击 Enable 模块，弹出如图 7.22 所示对话框，在 States when enabling 下拉列表框中选择 held 选项。

(7) Enabled and Triggered Subsystem2 模块和 Enabled and Triggered Subsystem3 模块参数设置：与步骤(6)基本相同，只在图 7.20 所示对话框中的 Trigger Type 下拉列表框中选择 falling 选项。

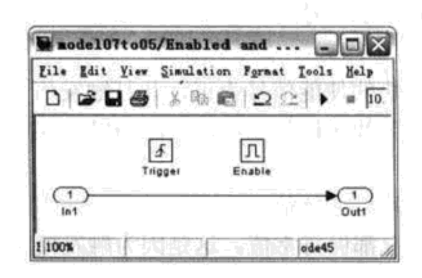


图 7.20　触发使能子系统模型

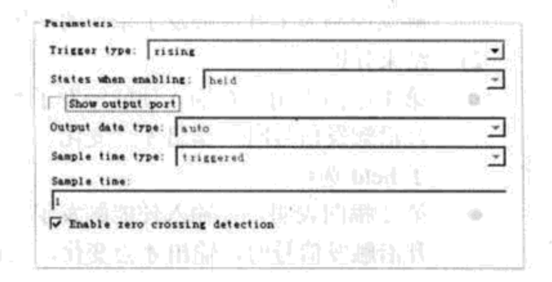


图 7.21　Triggered 参数设置

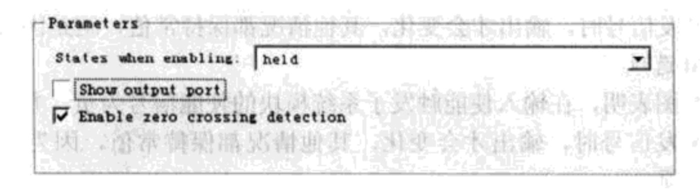


图 7.22　Enabled 参数设置

2. 进行仿真

模型系统参数采用默认值，仿真结果如图 7.23 所示。

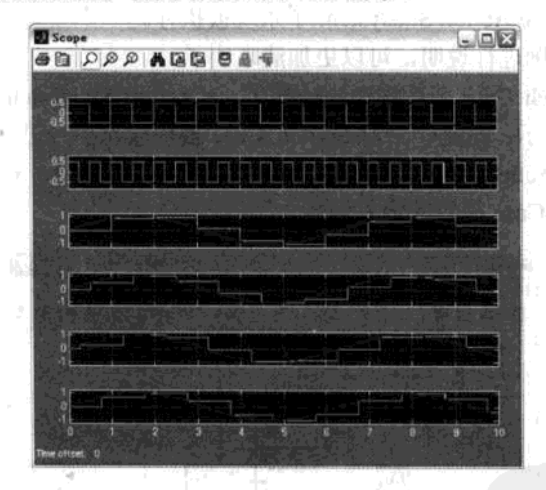


图 7.23　触发使能子系统仿真结果图

3. 结果分析

1)　子系统设置分析

- 第 1 幅图是使能控制信号。
- 第 2 幅图是触发控制信号。
- 第 3 幅是 Pulse Generator 模块和 Constant 求和信号直接输入使能触发模块，并且触发控制为上升沿触发子系统的结果图和正弦信号图。
- 第 4 幅是 Pulse Generator 模块和 Constant 求和信号取反输入使能触发模块，并且

触发控制为上升沿触发子系统的结果图和正弦信号图。

2) 结果分析

- 第 1 幅图表明，在输入使能触发子系统模块的使能信号为正，并且触发信号为上升沿触发信号时，输出才会变化，其他情况都保持常值，这是因为触发模块采用了 held 选项。
- 第 2 幅图表明，在输入使能触发子系统模块的使能信号为负，并且触发信号为上升沿触发信号时，输出才会变化，其他情况都保持常值，这是因为触发模块采用了 held 选项。
- 第 3 幅图表明，在输入使能触发子系统模块的使能信号为正，并且触发信号为下降沿触发信号时，输出才会变化，其他情况都保持常值，这是因为触发模块采用了 held 选项。
- 第 4 幅图表明，在输入使能触发子系统模块的使能信号为负，并且触发信号为下降沿触发信号时，输出才会变化，其他情况都保持常值，因为触发模块采用了 held 选项。

7.2.4 Switch Case 和 Switch Case Action Subsystem 子系统

Switch Case 子系统就是在对输入信号进行判断的基础上，然后选择相应的输出端口，使得相应的 Switch Case Action Subsystem 子系统被执行。

下面通过一个实例进行说明，可以更加清晰明了。

【例 0704】Switch Case 子系统和 Switch Case Action Subsystem 子系统建模实例演示建立模型如图 7.24 所示，这是一个对 Switch Case 模块输入信号进行端口选择，然后执行相应的 Switch Case Action Subsystem 子系统，输入信号为 1，2，3，4。其中 4 个 Step 模块是用来构造 Switch Case 模块的输入信号。

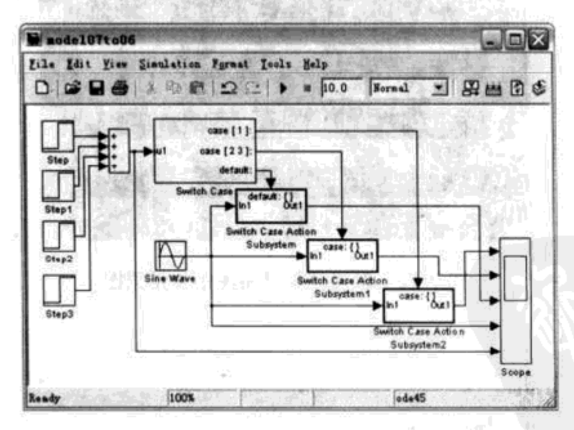


图 7.24　Switch Case 子系统和 Switch Case Action Subsystem 子系统建模

1. 对模块参数进行设置

(1) Step 模块设置。双击 Step 模块参数对话框，如图 7.25 所示，其中 Step time 为 0，Initial value 为 0，Final value 为 1，Sample time 为 0。

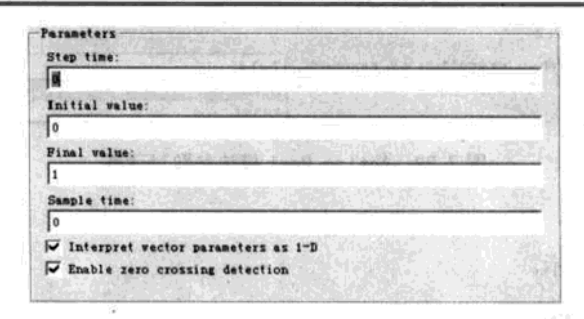


图 7.25　Step 模块参数对话框

- Step1 模块参置。其中 Step time 为 2，Initial value 为 0，Final value 为 1，Sample time 为 0。
- Step2 模块参置。其中 Step time 为 4，Initial value 为 0，Final value 为 1，Sample time 为 0。
- Step3 模块参置。其中 Step time 为 6，Initial value 为 0，Final value 为 1，Sample time 为 0。

(2) Sine Wave 模块设置。其中 Amplitude 为 1，Bias 为 0，Frequency 为 2， Phase Delay 为 0，Sample time 为 0。

(3) Switch Case 模块设置。双击 Switch Case 模块参数对话框，如图 7.26 所示，其中 Case condition 为{1,[2,3]}。

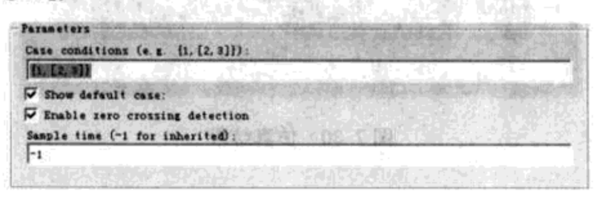


图 7.26　Switch Case 模块参数对话框

(4) Switch Case Action Subsystem 模块和 Switch Case Action Subsystem1/2 模块参数设置。在与不同的 Switch Case 模块端口连接，显示不会一样，可对比 Switch Case Action Subsystem 模块和 Switch Case Action Subsystem1 模块。双击 Switch Case Action Subsystem 模块，显示如图 7.27 所示。双击 Switch Case Action Subsystem1/2 模块弹出如图 7.28 所示。双击图 7.27 中 Action Port 模块，弹出参数对话框，如图 7.29 所示，在此不改变默认参数。

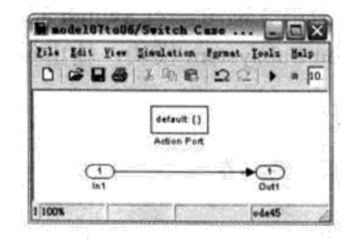


图 7.27　Switch Case Action Subsystem 模块窗口



图 7.28　Switch Case Action Subsystem1/2 模块窗口

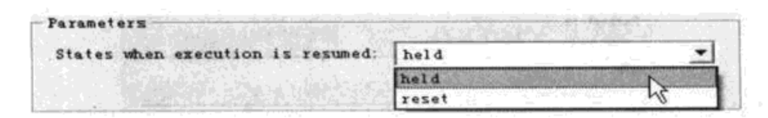


图 7.29 Action Port 模块参数对话框

2. 运行结果

结果如图 7.30 所示。

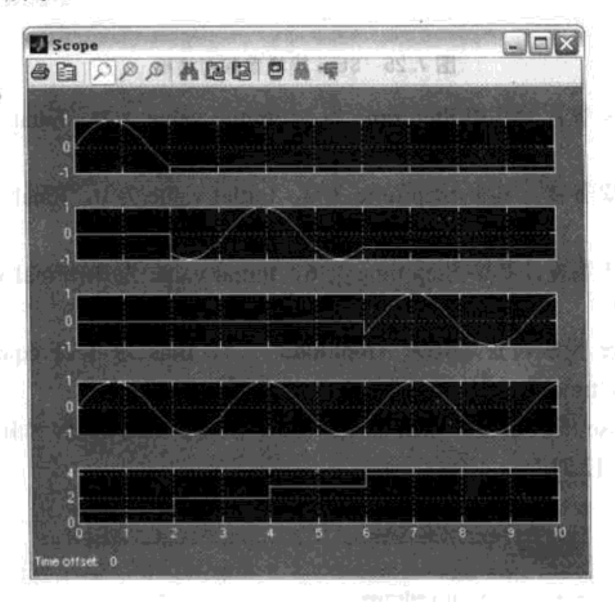


图 7.30 仿真结果

3. 结果分析

- 第 1 幅图是相对于 Switch Case 模块 Case 1 端口的结果。
- 第 2 幅图是相对于 Switch Case 模块 Case 2 端口的结果。
- 第 3 幅图是相对于 Switch Case 模块 Default 端口的结果。
- 第 4 幅图是 Sine Wave 模块信号。第五幅图是 Switch Case 模块的输入信号。

可以看出，在 Switch Case 模块输入信号为 1 时，这正好对应于 Case 1 情况，则 Switch Case Action Subsystem2 模块执行输出结果为第 1 幅图。当输入信号为 2 或 3 时，Switch Case Action Subsystem1 模块执行，输出结果为第 2 幅图。当 Switch Case 模块为 4 时，则执行 Default 端口连接的 Switch Case Action Subsystem 模块，输出结果如第 3 幅图。

7.3 Simulink 精装子系统

在前面介绍过如果创建一个子系统，可以方便模型的仿真和分析。然而仿真以前，必须对子系统中的模块进行初始化，在特殊情况下，一个大系统构造成的子系统，往往需要

不断地调整参数，如果每次都是打开子系统窗口进行参数设置，必将为仿真和分析带来很多麻烦。

在模型中，子系统的功能就相当于一个普通的模块，具有特定的输入端口和输出端口。

7.3.1 封装子系统

在这里要分清楚封装子系统和建立子系统是不同的两个概念，分别介绍如下：

建立子系统是将一组完成相关功能的模块包含到一个子系统当中，用一个模块来表示，主要是为了简化 Simulink 模型，增强 Simulink 模型的可读性，便于我们仿真和分析。在仿真前，需要打开子系统模型窗口，对其中的每个模块分别进行参数设置。虽然增强了 Simulink 模型的可读性，但并没有简化模型的参数设置。当模型中用到多个这样的子系统，但是每个子系统中模块的参数设置都不相同时，这就显得很不方便，而且容易出错。

为了解决简单建立子系统的不足，我们可以对子系统进行封装。将完成特定功能的相关模块集合在一起，对其中经常要设置的参数设置为变量，然后封装，使得其中变量可以在封装系统的参数设置对话框中统一进行设置。这就大大的简化了参数的设置，而且不容易出错。这非常有利于进行复杂的大系统仿真。

封装后的子系统可以作为用户的自定模块，作为普通模块一样添加到 Simulink 模型中应用，也可添加到模块库中以供调用。封装后的子系统可以定义自己的图标、参数和帮助文档，完全与 Simulink 其他普通模块一样。双击封装子系统模块，弹出对话框，进行参数设置，如果有任何问题，可以单击 help 按钮，不过这些帮助都是要创建者自己进行编写的。

下面对 Simulink 封装子系统的几个特点进行总结：

- 可以自定义封装子系统的图标。
- 双击封装后的子系统，弹出参数对话框，其中对话框是自定义。
- 封装子系统的帮助文档都是自定义编写的。
- 封装子系统有自己的工作区域。

以上特点为模型设计带来了很大的方便，具体如下：

- 将子系统作为一个黑匣子，用户不必了解其中的具体细节而直接使用。
- 将子系统中模块的参数设置统一到一个参数对话框，大大方便了系统的参数设置。
- 保护知识产权，防止篡改。

下面通过一个实际例子来说明如何封装一个子系统。

【例 0705】封装子系统。

(1) 建立模型，文件名为 model07to07.mdl，如图 7.31 所示，并对需要封装的模块进行框选。

(2) 生成子系统，右击选中的模块，在弹出菜单中选择 Create Subsystem 命令，并将文件另存为 model07to08.mdl，如图 7.32 所示。

(3) 封装子系统，右击生成的 Subsystem 模块，在弹出的菜单中选择 Mask Subsystem 命令，弹出如图 7.33 所示的对话框。在其中可以进行各种设置，这在后面进行介绍。

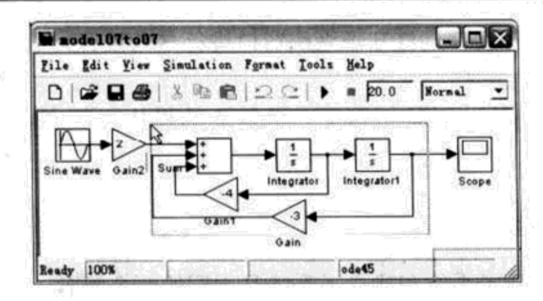


图 7.31 Simulink 模型图

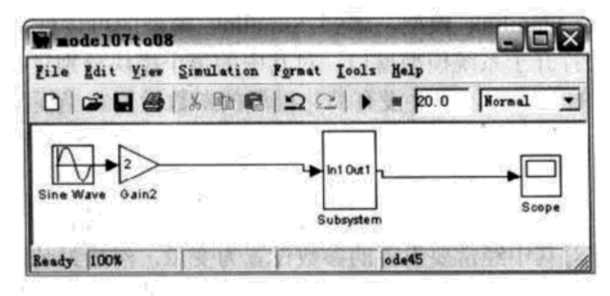


图 7.32 生成子系统



图 7.33 Mask editor 对话框

7.3.2 编辑封装子系统

Mask Editor 参数对话框可以创建和编辑封装子系统。右击生成的 Subsystem 模块，在弹出的菜单中选择 Mask Subsystem 命令，会弹出 Mask Editor 参数对话框，如图 7.33 所示。

Mask Editor 参数对话框包含了 4 个选项卡，每个选项卡都可以定义封装(mask)的一个特性。

(1) Icon 选项卡：允许定义模块图标。

(2) Parameters 选项卡：允许定义和描述封装对话框和参数对的字符变量。

(3) Initialization 选项卡：允许指定初始化命令。

(4) Documentation 选项卡：允许定义封装的类型，并且设定模块的描述和帮助。

(5) Mask Editor 参数对话框中最下面的 5 个按钮的功能。

- Unmask 按钮：解除封装，并关闭 Mask Editor 参数对话框，但是封装的信息仍然保留，为了能够恢复操作。为了恢复封装，右击选择的模块，在弹出的菜单中选择 Create Mask 命令。将弹出 Mask Editor 对话框，并显示以前的设定。当模型被关闭后其中的封装信息就被清除了。
- OK 按钮：应用所有设定，并关闭 Mask Editor 对话框。
- Cancel 按钮：关闭 Mask Editor 对话框，不应用所作的设定。
- Help 按钮：显示封装帮助文档。
- Apply 按钮：应用所作的参数设定，但是并不关闭 Mask Editor 对话框。

为了能够查看没有封装的子系统，可以右击子系统，然后在弹出的菜单中选择 Look Under Mask 命令，将会打开子系统，而且模块封装不会受影响。对每个选项卡的介绍如下：

1. Icon 选项卡

Icon 选项卡可以创建块图标，在图标中可包含描述性文字、状态方程、图像和图形，如图 7.33 所示。

1) Drawing commands 绘制命令

用户可以在 Drawing commands 区编写绘制块图标的命令。在 Simulink 中提供了一套命令，可以绘制文本、一个或多个图，可以显示传递函数，可以查看 Icon 选项卡中的在线帮助命令实例。用户只能通过这些命令来绘制图标。Simulink 就会按顺序执行命令区的命令，绘制命令有权调用所有封装工作空间中的变量。

下面这段命令演示了如何为 mx + b 封装子系统创建一个图标。

```
pos = get_param(gcb, 'Position');
width = pos(3) - pos(1); height = pos(4) - pos(2);
x = [0, width];
if (m >= 0), y = [0, (m*width)]; end
if (m < 0),  y = [height, (height + (m*width))]; end
```

这些初始化命令定义了绘制图形的数据，这些数据不受模块形状的影响而精确的绘制图标。而绘制图标的命令是 plot(x,y)。

2) Examples of drawing commands 绘制命令的实例

该选项组说明了不同的 Simulink 支持的图标绘制命令。为了能够了解其中命令的语法，可以从命令下拉菜单中选择相应的命令，Simulink 就会显示所选择命令的实例，并会在右下角产生一个图标。

3) Icon options 控制图标显示选项组

这个区域可以控制以下几个方面的显示：

- 框架(Frame)：图标框架是将模块封闭起来的矩形。用户可以通过选择 Frame 下拉列表框中的 Visible 或者 Invisible 选项来决定显示还是隐藏框架。默认情况是

显示框架。例如，下面 AND 模块的框架显示与隐藏，如图 7.34 所示。

- 透明(Transparency)：可以将图标设置成为不透明的或者透明的，也就是说是否显示图标下面的内容。默认设置是不透明，即覆盖 Simulink 绘制出来的信息，如端口标签。例如，如图 7.35 所示就是对 AND 模块设置为不透明和透明的图标，可以注意图标上名的文字。

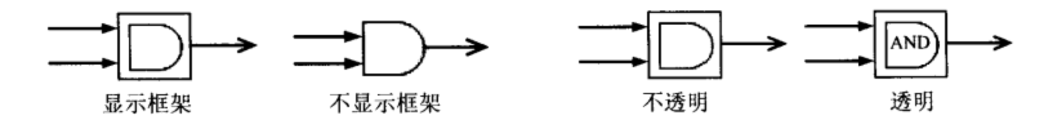


图 7.34　显示框架与不显示框架　　　图 7.35　模块图标透明与不透明

- 旋转(Rotation)：当模块旋转或者空翻(就是上下端口不变，左右端口变化)时，可以设定图标是否也跟着变化还是保持方向不变。默认情况下是不随模块的旋转而变化。例如，对 AND 模块进行旋转时，设定为固定(Fixed)或者旋转(Rotates)，如图 7.36 所示。

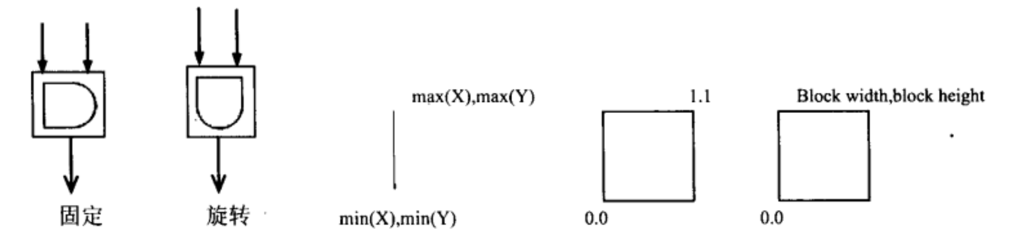


图 7.36　模块图标固定与旋转　　　图 7.37　图标尺寸

- 单位(Units)：这个选项控制了绘制命令坐标系统的单位，仅对 plot 和 text 绘制命令有效，可以选择 3 个选项之一：Autoscale，Normalized 和 Pixel，如图 7.37 所示。
    - ◆ Autoscale：可以根据框架来适当的调整图标的大小，当模块调整大小时，图标也同时调整大小。例如，如图 7.38 所示的图形用到的向量为：
    X = [0 2 3 4 9]; Y = [4 6 3 5 8]。
    - ◆ Normalized：在模块框架内绘制图标，左下角坐标是(0,0)，右上角坐标是(1,1)，坐标 X 和 Y 只能在 0～1 之间取值，当模块调整大小时，图标也同时调整大小。例如，如图 7.39 所示的图形用到的向量为：
    X = [.0 .2 .3 .4 .9]; Y = [.4 .6 .3 .5 .8]。
    - ◆ Pixel：用 X 和 Y 的像素值来绘制图标，这种方法绘制图形的方法，在模块进行大小调整时，图标不会改变大小。如果要强制随模块的变化来调整大小时，必须根据模块的大小来定义绘制命令。

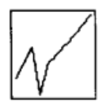




图 7.38　Autoscale 绘制的图标



图 7.39　Normalized 绘制的图标

2. Parameters 选项卡

Parameters 选项卡可以创建和修改封装子系统的参数，这些参数决定了封装子系统的各种特性和行为，如图 7.40 所示。

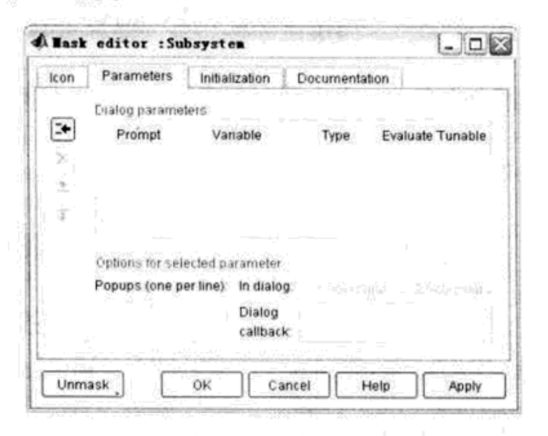


图 7.40　Parameters 选项卡

Parameters 选项卡包括两个选项组：

- Dialog parameters 选项组：可以选择和改变一些主要封装子系统的特性。
- Options for selected parameter 选项组：可以为 Dialog parameters 选项组中设定的参数添加附加的选项。

Parameters 选项卡的左端按钮允许用户添加、删除封装子系统的参数和改变这些参数的位置。

1)　Dialog parameters 控制面板

以表格的形式列举封装子系统的参数。每一行显示一个封装子系统参数的主要特征。例如，进行如图 7.41 所示设置，然后单击 OK 按钮，最后双击封装子系统 Subsystem 模块，就会弹出如图 7.42 所示的对话框。

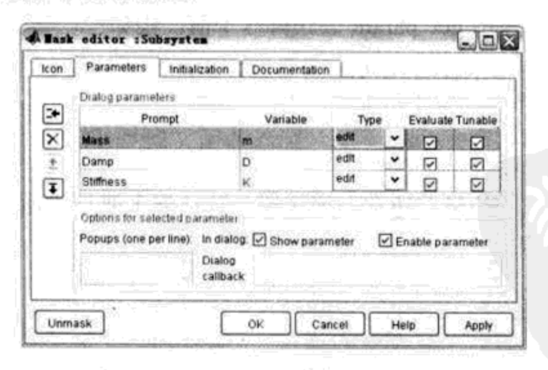


图 7.41　Parameters 标签列表框设置

下面对图 7.41 所示的 Parameters 选项卡中的参数进行说明：

- Type：指编辑参数的方式，如图 7.43 所示。包括 edit，checkbox 和 popup 形式，分别为文本输入、多选和下拉菜单形式。

- Evaluate：如果选中该复选框，表示在将用户输入的表达式赋予相应的变量之前对表达式求值。否则 Simulink 就将表达式作为字符值赋予变量。例如，在用户在编辑框中输入 gain，而且 Evaluate 是选中时，那么 Simulink 就会先求 gain 的值，然后赋给相应的变量。假如没有选中 Evaluate，则直接给字符串 'gain' 赋予变量。
- Tunable：该选项允许在 Simulink 进行仿真过程中改变封装子系统中的参数。

图 7.42　设置子系统 Subsystem 模块参数

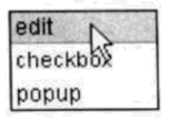


图 7.43　Type 参数内容

2) Options for selected parameter 选项卡

该选项卡允许用户向 Dialog parameters 表中的参数添加附加的选项。

- Show parameter 复选框：只有选中了，所选择的参数才会出现在封装子系统模块的参数对话框中。
- Enable parameter 复选框：不选中该复选框，所选择的参数在封装子系统模块的参数对话框中是不可编辑的。例如，当选择 Mass 菜单选项，然后不选中该复选框，单击 OK 按钮，最后双击封装子系统模块，就会弹出参数对话框如图 7.44 所示，注意其中的 Mass 参数是不可编辑的。
- Popup：仅当 Dialog Parameters 控制面板中的编辑方式 Type 为 pop-up 时，才会处于可编辑状态。输入相应的 pop-up 菜单选项，一行一个菜单命令。
- Callback(回调)：在用户编辑所选择的菜单时，才允许 Simulink 执行所输入的代码。回调能够在模块的工作范围内创建和引用变量。如果回调需要封装子系统的参数值，可以使用 get_param 去获得这个值。例如：

```
if str2num(get_param(gcb, 'g'))<0
   error('Gain is negative.')
end
```

3) 参数增减控制按钮

在 Parameters 选项卡的最左边有一列按钮，其功能主要是增加、删除参数和对参数进行排序。如图 7.45 所示。

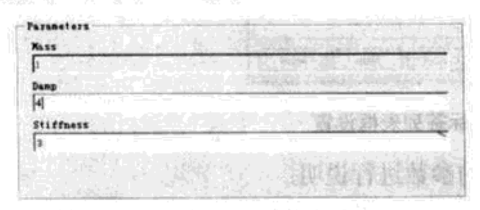


图 7.44　Mass 不能编辑的子系统参数对话框



图 7.45　参数增减控制按钮

3. Initialization 选项卡

Initialization 选项卡允许用户输入 MATLAB 命令来初始化封装子系统。Initialization 选项卡如图 7.46 所示。

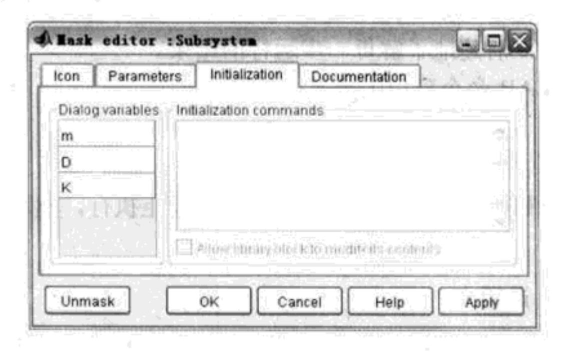


图 7.46　Initialization 选项卡

Simulink 在以下几种情况下执行初始化命令：

- 装载模型。
- 开始仿真和更新模块图形(选择 Edit | Update Diagram 命令)。
- 旋转模块。
- 重画模块图标(如果封装子系统的图标代码中的依靠 Initialization 选项卡中初始化的变量)。

1) Initialization 选项卡

包括以下几个控制选项：

- Dialog variables 选项组：此列表中显示了与封装子系统参数相关的变量名。用户可以从这个列表中复制参数名移到 Initialization commands 框中。也可以使用这个列表来更改参数变量，用鼠标双击相应的变量就可以更改了，然后按 Enter 键确定。
- Initialization commands 选项组：在 Initialization commands 中输入初始化命令，可以是任何的 MATLAB 表达式，例如，MATLAB 函数，运算符和在封装模块空间中的变量，但是初始化命令不能够是基本工作空间的变量。初始化命令要用分号来结尾，避免在 MATLAB 命令窗口中出现回调结果。
- Allow library block to modify its contents 复选框：该复选框仅当封装子系统存在于模块库中才可用。选中这个复选框允许模块的初始化代码修改封装子系统的内容。例如可以允许初始化代码增加和删除模块，还可以设置模块的参数。否则，当试图通过模块库中的模块修改模块中的内容时，Simulink 仿真就会出现错误。不过这个还可以在 MATLAB 命令窗口中实现，选中要修改内部模块的封装子系统模块，然后在命令窗口中输入如下命令：

```
>> set_param(gcb, 'MaskSelfModifiable', 'on');
```

然后保存这个模块。

2) 调试初始化命令

用户可以通过以下几种方法来调试初始化命令。

- 在命令的结尾不是用分号，以便能够在 MATLAB 命令窗口中能够直接查看相关命令运行结果。
- 在命令中间设定一些键盘控制命令，如中断、键盘输入参数等，可以实现人机交互，这样就可以清楚地了解每一步运行的结果。
- 可以在 MATLAB 命令窗口中输入以下命令：

```
>> dbstop if error
>> dbstop if warning
```

这些命令可以使当初始化命令发生错误时，就会停止执行，然后用户就可以检查封装子系统的工作空间。

4. Documentation 选项卡

Documentation 选项卡允许用户定义或修改类型、描述以及封装子系统模块的帮助文档。下面就是 Mx"+Dx'+Kx 的一个实际例子，如图 7.47 所示，然后单击 OK 按钮，最后双击封装子系统模块，弹出对话框如图 7.48 所示。

1) 仔细观察图 7.47 和图 7.48，就会发现图 7.46 中各个参数的作用。

- Mask type：为封装子系统模块参数对话框设置说明的标题，仅仅用来对文档进行分类，它出现在模块参数对话框中和所有的模块封装编辑窗中。用户可以选择喜欢的名称，当 Simulink 创建模块对话框时，它会自动在后面添加“(mask)”，以区别系统内建的模块。
- Mask description：对封装子系统模块的工作进行说明，可以在说明中使用 Enter 和 Space 键。此参数中要尽量对模块进行描述，以便用于其他模型当中。
- Mask help：该模块的帮助文档，可以单击封装子系统中的 Help 按钮进行查看，这个可以讲述如何对模块的参数进行设置，以便用于其他模型当中。

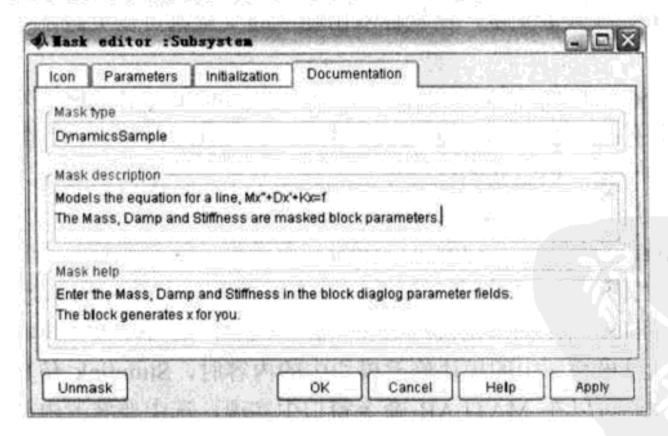


图 7.47　Documentation 标签列表框

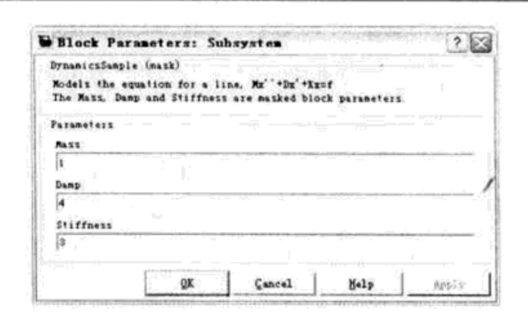


图 7.48 设置好 Documentation 窗选项的对话框

2) 除了使用这个参数来设定模块帮助以外，还可以使用用户自己编写的其他文档，可以使用的文档有以下几种：

- 超链接(如以 http:，www，file:，ftp:和 mailto:开头的字符)。
- 网络命令(调用浏览器)。
- eval 命令(求取 MATLAB 字符值)。
- 用 HTML-tagged 文本来显示网络浏览器。

Simulink 检查封装子系统模块帮助文档的第一行，如果其中有超链接，如：

http://www.SciEi.com 或者 file:///c:/mydir/helpdoc.html

Simulink 将在浏览器中显示这些内容。

如果 Simulink 检查到有网络命令，例如：

```
web([docroot '/My Blockset Doc/' get_param(gcb,'MaskType')'.html'])
```

或者有 eval 命令，例如：

```
eval('!Word My_Spec.doc')
```

Simulink 就会执行这些命令。否则，Simulink 只会显示模块帮助域中的内容。

7.3.3 联系封装子系统的参数与子系统中的模块参数

在 7.3.2 节中已经讲解了如何设定封装子系统的参数，但是这些参数并没有和子系统内部的模块连接起来。也就是说，即使封装子系统的参数设定完成后，封装子系统模块还是不能够使用，不能和内部的模块进行数据传输。

为了能够使得封装子系统中设置的参数能够被子系统内部的模块所使用，必须将它们连接起来。这就能够让用户使用封装子系统的参数去设置子系统内部的模块参数。

为了能够连接这些参数，首先打开模块参数对话框，并在参数对话框中输入所需要的表达式，表达式中的变量都来自于封装子系统所设置的参数。如前面的 Mx”+Dx’+Kx 封装子系统的例子，就是使用这种方法来确定内部的 M, D 和 K 等参数。用户可以用封装子系统模块的初始化代码将封装子系统的参数间接地与模块参数连接起来，通过这种方法，初始化代码在封装子系统工作空间创建变量，这些变量的值就是封装子系统的参数的函数，这样就可以通过设置封装子系统参数来获取变量的值，然后通过变量传递给封装子系统内部的模块。

7.4 精装子系统实例

为了能够对精装子系统有一个充分的了解，这里特别增加一章，在此不讲述各个模块和菜单的功能，这里主要集中完成一个实际的例子，叙述封装精装子系统的操作步骤。

【例 0706】建立一个单自由度系统的动力学，对其进行封装，通过封装子系统模块的参数对话框来设置系统的质量、阻尼和刚度，从而达到方便计算和仿真的效果。

操作步骤如下：

(1) 建立系统的 Simulink 动力学仿真模型，如图 7.49 所示。模型中的 Gain 和 Gain1 模块都是由参数组成，Gain 为-K/M，Gain 为-D/M。其他模块的参数设置采用模块值。

(2) 创建子系统模型，如图 7.50 所示。选定需要封装的部分，右击选择的模块，从菜单中选择 Create Subsystem 命令，建立的子系统模型如图 7.50 所示，另存文件为 model07to10.mdl；然后右击创建的子系统模块，在弹出的菜单中选择 Mask Subsystem 命令。

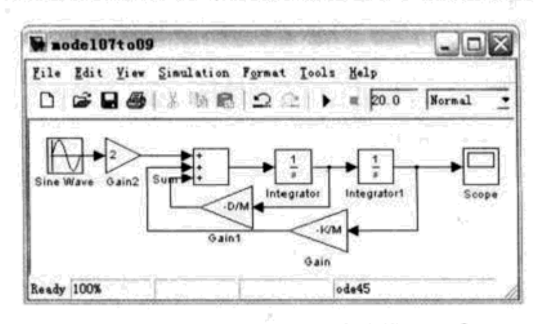


图 7.49 系统动力学仿真模型

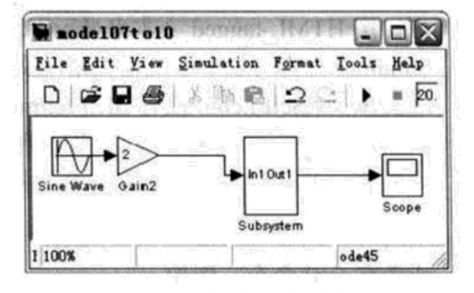


图 7.50 创建子系统模型

(3) 创建封装子系统模块图标。右击子系统模块，在弹出的菜单中选择 Edit Mask 命令。在 Mask editor 对话框中，选择 Icon 选项卡，在 Drawing commands 框中输入如下代码：

```
plot([0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5],[1.0000 -0.4859 0.1044 0.0773 -
0.1136 0.0819 -0.0378 0.0066 0.0075 -0.0097 0.0067]);
```

并对 Icon options 进行设置，如图 7.51 所示，单击 OK 按钮，将会得到结果如图 7.52 所示图标。

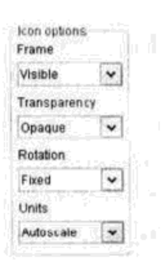


图 7.51 Icon options 设置

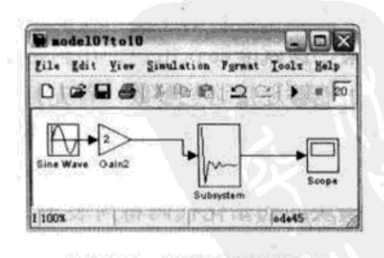


图 7.52 创建子系统图标

(4) 设置封装子系统参数。右击子系统模块，在弹出的菜单中选择 Edit Mask 命令。在 Mask editor 对话框中选择 Parameters 选项卡，增加 3 个参数分别为 Mass, Damp 和

Stiffness，其对应的变量名分别为 M,D 和 K，设置如图 7.53 所示。不过如果此时双击子系统模块，在弹出的对话框中没有子系统模块的功能说明和参数设置帮助。

图 7.53 Parameters 参数设置

(5) 初始化设置。在 Mask editor 对话框中选择 Initialization 选项卡，并在 Initialization commands 框中输入如下代码：

```
M=1;
D=4;
K=3;
```

这就使得不用对子系统进行参数设置就可以使用默认参数运行 Simulink 模型了。

(6) 设置模块帮助和说明。在 Mask editor 对话框中选择 Documentation 选项卡，在编辑框中输入的说明分别如下：

```
Mask type:
Dynamics example:
Mask description:
This is a dynamics example about singular degree of freedom.
Mx''+Dx'+Kx=f
You can set the parameters through the subsystem dialog.
Mask help:
You can input the Parameters M, D, K in the subsystem dialog.
Different M, D, K represent different system.
```

结果如图 7.54 所示。

(7) 封装子系统。设置完成后单击 OK 按钮，此时就可以运行 Simulink 模型了。

说明： 可能有人会奇怪为什么不设置子系统参数就可以运行模型了，其实在初始化参数设置中就设定了默认的参数，如果用户不对子系统参数进行设置，就表示用默认的参数进行仿真，并得到相应的结果。当然用户也可以进行自己的设置。

(8) 设置参数。双击分装子系统模块，在弹出的对话框中设置如下：

```
Mass:1;
Damp:2;
Stiffness:3;
```

如图 7.55 所示，可以看到用户刚才设置的模块类型和模块说明，单击 OK 按钮。

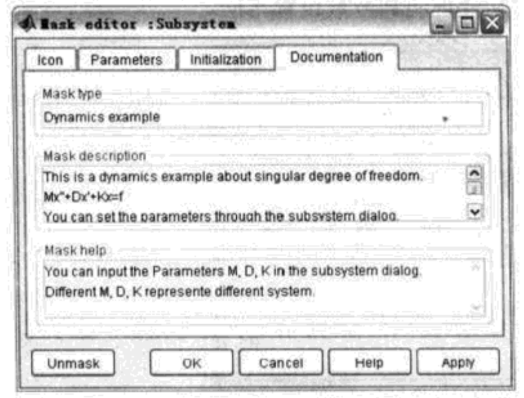


图 7.54 子系统模块帮助与说明设置

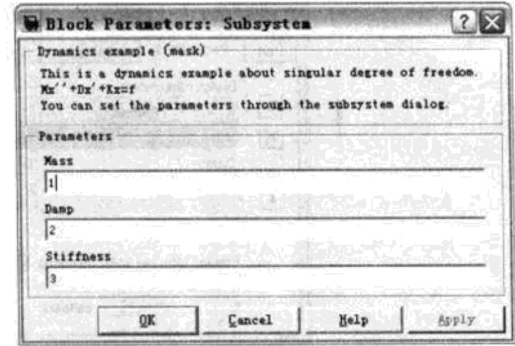


图 7.55 子系统模块参数设置

(9) 查看相关帮助。在图 7.55 参数设置对话框中单击 Help 按钮，就可以看到开始时设置的帮助文档。如图 7.56 所示。

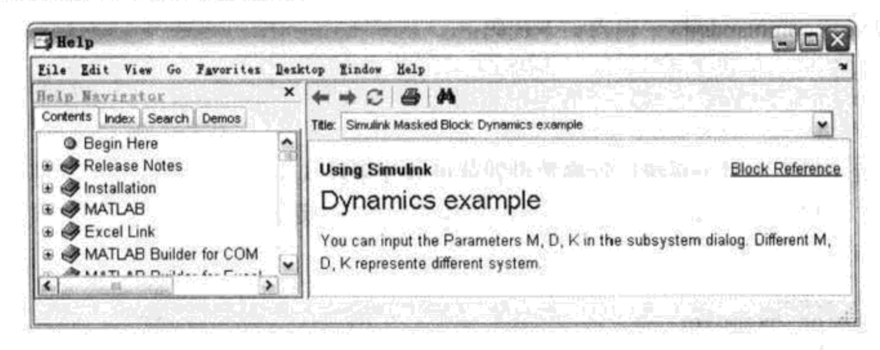


图 7.56 子系统模块帮助

(10) 运行仿真并查看结果。保存 Simulink 模型，然后在 Simulink 窗口中单击【运行】按钮，然后双击 Scope 模块查看仿真结果。结果如图 7.57 所示。

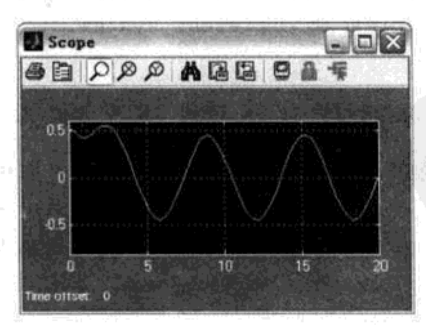


图 7.57 仿真结果

7.5　Simulink 模块库技术

前面的章节系统地介绍了 Simulink 子系统的封装技术。子系统的封装给不同领域的设计者带来了很大的方便，使用者调用这些模块如同调用 Simulink 内部模块一样，可以使用封装子系统的帮助来了解如何进行模块设置，而不需要了解模块内部的构成。

但是如果封装的子系统比较多，应用的范围也不相同，如何有效地管理这些模块就成为一个非常重要的问题。

7.5.1　模块库

模块库就是指具有某种属性的一类模块的集合。在 Simulink 模块浏览器中有大量的模块库，用户就可以调用其中的模块来进行各种系统的仿真。

Simulink 允许用户自己开发模块，并在建立自己模块库的同时，对开发的模块进行有效地管理。自定义模块库的使用方法和 Simulink 其他自带的模块库的使用完全一样。

- 模块库：某一类模块的集合。
- 库模块：模块库中的模块。
- 引用块：调用模块库中的模块。
- 关联：引用块与对应模块库中的模块建立关联，以便在模块库中模块发生改变时，Simulink 模块库中的模块会自动更新。

7.5.2　建立模块库

在使用模块库之前，首先要创建模块库，其操作步骤如下：

(1) 在 Simulink Library Browser 窗口中选择 File | New | library 命令，如图 7.58 所示。

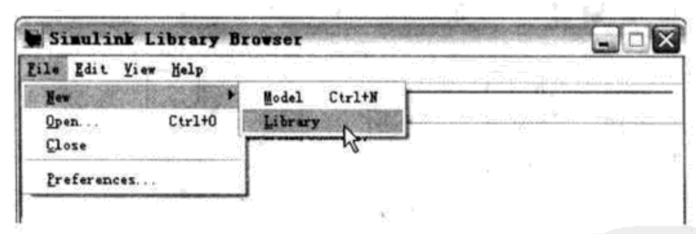


图 7.58　新建模块库菜单

(2) 将用户自定义的模块或是其他模块库中的模块拖放到新建的模块库中。

(3) 保存模块库。

【例 0707】创建模块库和使用模块库。

将上一节中创建好的模块拖放到新建的模块库中，如图 7.59 所示。保存为 model07to11.mdl。然后就像使用普通模块一样使用。例如建立如图 7.60 所示的模型图，文件名为 model07to12.mdl，单击【运行】按钮。




图 7.59 新建模块库

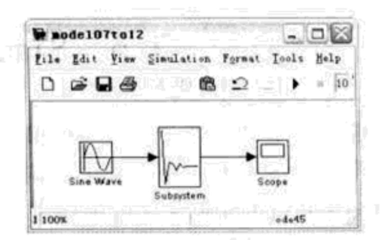


图 7.60 自建模块实例模型

如果需要对引用块的内容进行修改，主要有以下两种修改方式：

- 用鼠标右击模块，在弹出的菜单中选择 Link Options | Go to library block 命令，对引用模块相应的模块库进行解锁，然后选择 Edit | Update diagram 命令；
- 用鼠标右击模块，在弹出的菜单中选择 Link Options | Disable link 命令，然后用鼠标右击，在弹出菜单中选择 Look under mask 命令。

7.5.3 库模块与引用块的关联

在上一节例子中的模型窗口中的库模块是不能够编辑的。Simulink 通过关联来表示模块库中的模块与相应的应用块之间的关系，为了使用户对关联有一个比较深刻的了解，通过一个具体的例子来说明。

【例 0708】关联的使用方法

(1) 如图 7.61 所示，新建模型文件 model07to13.mdl，在此模型中建立两个完全相同的系统，Subsystem 和 Subsystem1 模块都是来自自建的模块库中。

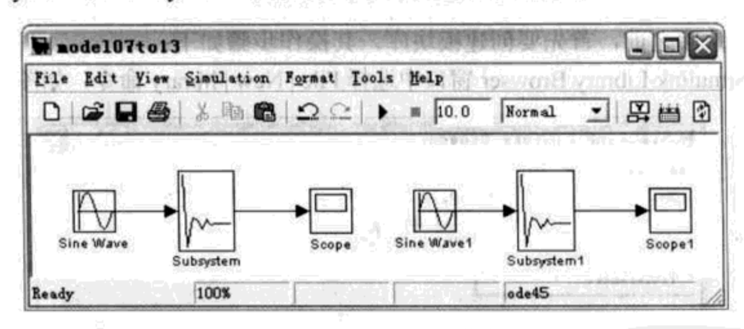


图 7.61 实例 0708 系统模型

(2) 右击模型中的 Subsystem 模块，在弹出的对话框中选择 Link Options | Disable link 命令。

(3) 右击模块库中的模块，在弹出的菜单中选择 Look under mask 命令，编辑如图 7.62 所示，添加一行模型说明后保存。

(4) 用鼠标分别右击模型中的 Subsystem 模块和 Subsystem1 模块，在弹出的菜单中选择 Look under mask 命令，可以对比弹出窗口的不同，如图 7.63 所示。

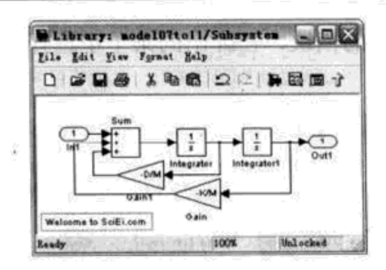


图 7.62　编辑库模块

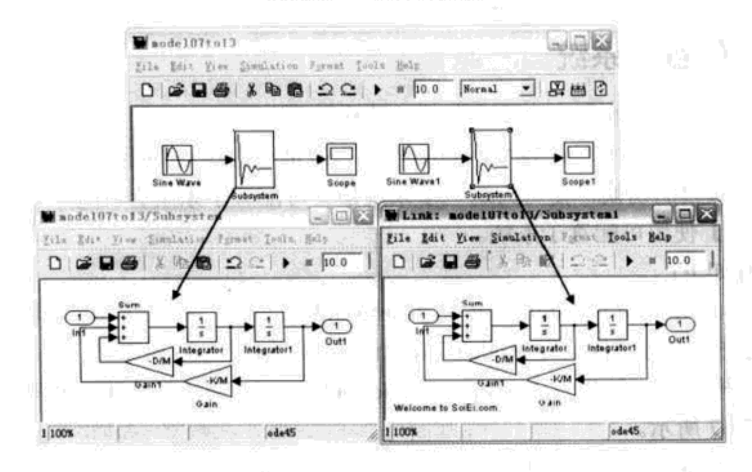


图 7.63　比较关联与不关联两个模块的不同

从图 7.63 中可以发现，Subsystem 模块和 Subsystem1 模块的不同，Subsystem 模块并没有随着库模块一起改变，而 Subsystem1 模块则随着库模块一起改变。从中就可以体会到关联的作用了。

(5) 通过 Subsystem 模块来修改模块库中的模块。例如，分别用鼠标右击模型中的 Subsystem 模块和 Subsystem1 模块，在弹出的菜单中选择 Look under mask 命令，弹出窗口中添加一行文本，如图 7.64 所示，然后保存。模型说明不要和前面相同，否则看不出效果。

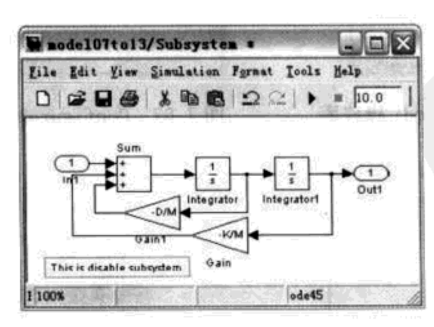


图 7.64　Subsystem 模块子系统

(6) 右击模型中的 Subsystem 模块，在弹出的菜单中选择 Link options | Restore link 命令，会弹出一个对话框，如图 7.65 所示。然后单击 Update Library 按钮。

(7) 用鼠标右击模块库中的模块，在弹出的菜单中选择 Look under mask 命令，会发现库模块也随之改变了。

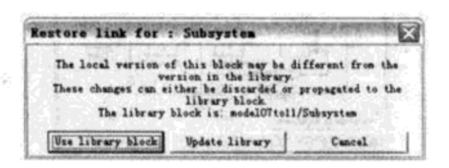


图 7.65 Restore link 对话框

7.5.4 可配置子系统

可配置子系统只能在用户自定义的模块库中使用。

在模型库中经常会出现很多个不同的模块，但是它们却具有相同的功能，如果用户需要在它们之间频繁的切换，便可为这些子系统建立可配置子系统。

【例 0709】使用可配置子系统。

具体步骤如下：

(1) 复制模块 Subsystem，并从 Port&Subsystems 模块库中拖放 Configurable Subsystem 模块到自定义模块库 model07to11.mdl 中，如图 7.66 所示。

(2) 保存模块库，然后双击 Configurable Subsystem 模块，确定要相互切换的自定义模块，如图 7.67 所示。

(3) 将 Configurable Subsystem 模块拖放到 Simulink 模型中，如图 7.68 所示。

(4) 在 Simulink 模块窗口，选择 Edit | Block Choice 命令，如图 7.69 所示。

Simulink 的模块库技术为用户提供了非常大的空间，可以很好的组织管理自定义的模块。不同领域的用户可以建立自己独有的模块库，便于系统开发的维护和修改。

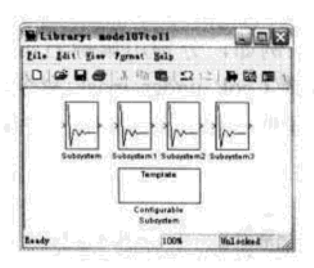


图 7.66 model07to11.mdl 模块库

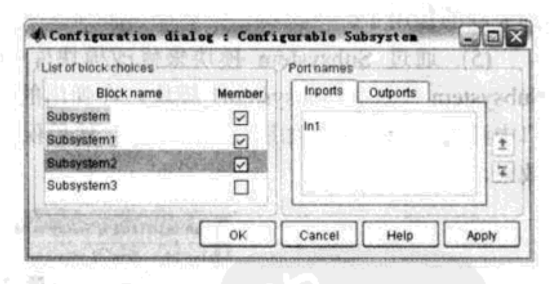


图 7.67 Configurable Subsystem 模块参数对话框

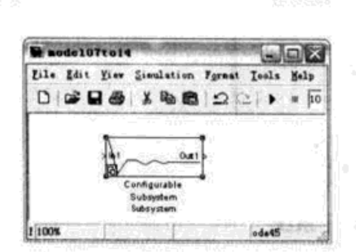


图 7.68 model07to14 模型图



图 7.69 Edit | Block Choice 菜单选项

第 8 章　Simulink 数值计算

Simulink 仿真是一种基于数值计算的仿真，在仿真过程中同样会遇到计算精度的问题，对待不同性态的方程，必须用不同的求解方法。

本章介绍了微分求解的种类以及不同的适用范围。介绍了 Simulink 对待刚性方程所采用的方法以及 Simulink 模型中存在代数环时，可以采取的解决办法。

本章主要内容包括：

- 微分方程求解器 Solver
- 刚性方程求解实例
- Simulink 仿真中的代数环问题

8.1　微分方程求解器 Solver

由 Simulink 所提供的求解器都是当今国际上数值计算研究的最新成果，采用的都是速度最快，精度最高的计算方法。即便如此，也没有一个万能的计算方法，能够非常理想地求解各类微分方程。不同的系统需要利用不同的求解器，故了解系统的特性是非常重要的，如系统方程是否是刚性方程(stiff equation)等。下面来具体介绍各种计算方法，这对于不同的系统选择不同的方法是至关重要的。

1. ode45

这种求解器采用 Runge-Kutta 方法，这也是利用 Simulink 求解微分方程时最常用的一种方法。这种算法精度适中，是在计算方针中的首选项。

它是利用有限项的 Taylor 级数取近似解函数，而误差的来源就是 Taylor 的截断项，误差就是截断误差。

ode45 分别采用 4 阶、5 阶 Taylor 级数计算每个积分步长终端的状态变量近似值，并利用这个级数的值相减，得到的误差作为计算误差的判断标准。如果误差估计值大于这个系统的设定值，那么就把该积分步长缩短，然后重新计算；如果误差远小于系统的设定值，那么就将积分步长放长。

2. ode23

这种求解器采用 Runge-Kutta 方法，为了能够达到 ode45 同样的精度，ode23 的积分步长总要比 ode45 取的小。因此，ode23 处理“中度 Stiff”问题的能力优于 ode45。

ode23 是利用有限项的 Taylor 级数取近似解函数，而误差的来源就是 Taylor 的截断项，其中，误差就是指截断误差。

ode45 分别采用 2 阶、3 阶 Taylor 级数计算每个积分步长终端的状态变量近似值，并利用这个级数的值相减，得到的误差作为计算误差的判断标准。如果误差估计值大于这个系统的设定值，那么就把该积分步长缩短，然后重新计算。如果误差远小于系统的设定

值，那么就将积分步长放长。

ode23 和 ode45 都是变步长算法。

3. ode113

ode113 与 ode45 和 ode23 不同，它采用的变阶 Adams 法是一种多步预报校正算法。

使用 ode113 的操作步骤如下：

(1) 在预报阶段，用一个(n-1)阶多项式近似导函数。

(2) 该预报多项式的系数可通过前面(n-1)个节点及其导数值确定。

(3) 用外推方法计算下一个节点。

(4) 在校正阶段，通过对前面 n 个解点和新的试探解点运用拟合技术获得校正多项式。

(5) 用该校正多项式重算试探解，即获得校正解。

(6) 用预报解和校正解之间的差作为误差，与系统设定值比较，用来调整积分步长，类似于 ode45 和 ode23 方法。

ode113 在执行过程中还自动地调整近似多项式的阶数，以平衡其精确性和有效性。

ode45 和 ode23 采用的是 Taylor 级数方法，而 ode113 采用的是多项式方法，计算导数的次数也比前面两种方法次数少，所以在计算光滑系统时，ode113 的速度更快。

4. ode15s

ode15s 是专门用来求解刚性(Stiff)方程的变阶多步算法，包含一种对系统动态转换进行检测的机理。这种检测使这一算法对非刚性(Stiff)系统计算效率低下，尤其是对那种有快速变化模式的系统情况。

5. ode23s

ode23s 同 ode15s 一样都是用来求解刚性方程的，是基于 Rosenbrok 公式建立起来的定阶单步算法。由于计算阶数不变，所以计算效率要比 ode15s 效率高。

6. ode23t

用来求解中度刚性方程。

7. ode23tb

用来求解刚性方程。

8.2　刚性方程求解实例

【例 0801】求解微分方程 $\ddot{x}+999\dot{x}+0.999=0$，初值为 $x(0)=1,\dot{x}(0)=0$。

通过这个刚性方程可以发现，求解器使用不当，产生的结果差别非常大。

首先求解方程的解析解，然后通过用不同求解器所得到的数值方法进行比较，来说明不同算法之间的不同，以及不合适方法所带来的错误结果。解题步骤如下：

(1) 利用 MATLAB 求解解析解。

```
>> sciei=dsolve('D2x+999*Dx+0.999*x=0','x(0)=1','Dx(0)=0','t')
sciei =
```

```
(1/2+15/499498*277221390^(1/2))*exp(3/100*(-16650+277221390^(1/2))*t)+(-
15/499498*277221390^(1/2)+1/2)*exp(-3/100*(16650+277221390^(1/2))*t)
>> aokee=diff(sciei,'t')
aokee =
(1/2+15/499498*277221390^(1/2))*(-999/2+3/100*277221390^(1/2)) *exp
(3/100*(-16650+277221390^(1/2))*t)+(-15/499498*277221390^(1/2)+1/2)*(-
999/2-3/100*277221390^(1/2))*exp(-3/100*(16650+277221390^(1/2))*t)
```

从 sciei 中可以看出，包含两项，将 sciei 进行简化，可得到如下结果：

$$\text{sciei} = e^{-0.001t} - 1.001\times10^{-6}e^{-998.9990t}$$

$$\text{aokee} = -0.001e^{-0.001t} + 0.001e^{-998.9990t}$$

从结果可以看出，sciei 中有两项，一个系数很小、衰减非常快项 $e^{-0.001t}$，另外一个为 $-1.001\times10^{-6}e^{-998.9990t}$，后面一个起主要作用。aokee 中同样有两项，一个系数很小、衰减非常快的 $-0.001e^{-0.001t}$，另外一个为 $0.001e^{-998.9990t}$ 。

(2) 建立 Simulink 模型，保存为 model08to01.mdl，如图 8.1 所示：

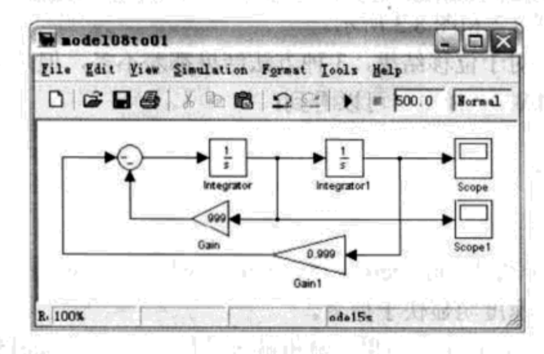


图 8.1　微分方程模型

(3) 为了能够更加清楚地看到 3 种方法的区别，可通过运行以下代码，保存文件为 mfun08to01.m。

```
%mfun08to01.m
%copyright@ www.sciei.com
%求解方程解析解，
sciei=dsolve('D2x+999*Dx+0.999*x=0','x(0)=1','Dx(0)=0','t'); %得到位移解表
达式
aokee=diff(sciei,'t'); %得到速度解析解表达式
t=0:0.1:500;
dx=eval(char(sciei)); %得到速度解析解数值
x=eval(char(aokee)); %得到位移解析解数值
%使用 ode45 求解器进行仿真
opts=simset('Solver','ode45'); %设置 ode45 为当前求解器
[t1,x1,s]=sim('odeexample01',500,opts);%进行模型仿真
%使用 ode15s 求解器进行仿真
opts=simset('Solver','ode15s'); %设置 ode15s 为当前求解器
[t2,x2,s]=sim('model08to01',500,opts);%进行模型仿真
%绘制位移计算结果
figure(1)
plot(t,x,'k'); %绘制解析解曲线
hold on
plot(t1,x1(:,1),'b:.','LineWidth',3); %绘制 ode45 仿真曲线
hold on
plot(t2,x2(:,1),'r-'); %绘制 ode15s 仿真曲线
axis([249.7 250 0.7785 0.7795])
```

```
legend('符号','ode45','ode15s') %添加标签
xlabel('时间') %添加横坐标
ylabel('位移') %添加纵坐标
hold off
%绘制速度计算结果
figure(2)
plot(t,dx,'k','LineWidth',3); %绘制解析解曲线
hold on
plot(t1,x1(:,2),'b:'); %绘制 ode45 仿真曲线
hold on
plot(t2,x2(:,2),'r-'); %绘制 ode15s 仿真曲线
axis([249.7 250 -7.793e-4 -7.777e-4])
legend('符号','ode45','ode15s') %添加标签
xlabel('时间') %添加横坐标
ylabel('速度') %添加纵坐标
hold off
numberode45=length(x1) %求 ode45 仿真变量的维数
numberode15s=length(x2) %求 ode15s 仿真变量的维数
```

(4) 仿真结果如图 8.2 和图 8.3 所示。

从图中可以看出，对于位移结果，3 种方法结果都差不多，但是 ode15s 的求解点个数远远小于 ode45。在 MATLAB 窗口可以看到：

```
numberode45 =
     150508
numberode15s =
    86
```

ode15s 的求解点个数为 numberode15s =86, ode45 的求解点个数为 numberode45 = 150508，可以看出前者速度明显快于后者。

从图 8.3 可以看出，ode45 的结果和解析解相差比较大，而 ode15s 比较理想，从中可以看出，如果选择不合适的求解器，不但费时而且结果不理想。

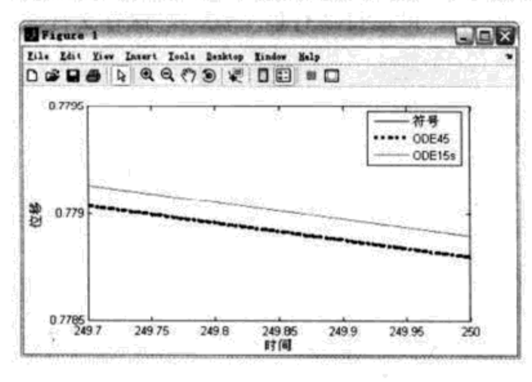




图 8.2　位移结果图

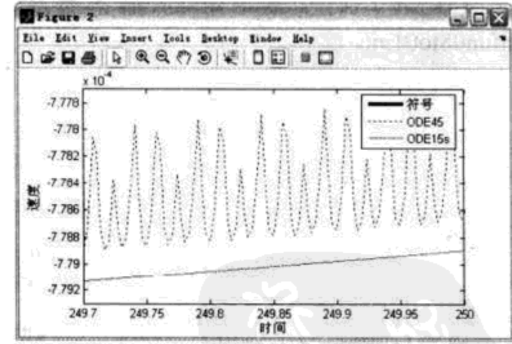


图 8.3　速度结果图

8.3　Simulink 仿真中的代数环问题

无论是采用 Fortran、C 还是 MATLAB 和 Simulink，在编写计算程序时，都会遇到代数环问题，而代数环会给计算程序带来很大的麻烦，需要特别注意。

本节就针对 Simulink 的代数环问题进行讨论。

【例 0802】利用 Simulink 求解方程组$\begin{cases}\dot{x}_1=-a_1x_2+a_2u\\x_2=a_3x_1+a_4\dot{x}_1\end{cases}$。

此题中，直接利用 Simulink 求解此方程时，会遇到代数环问题。

解题步骤如下：

(1) 用 Simulink 建立方程组模型，保存为 model08to02.mdl，如图 8.4 所示。

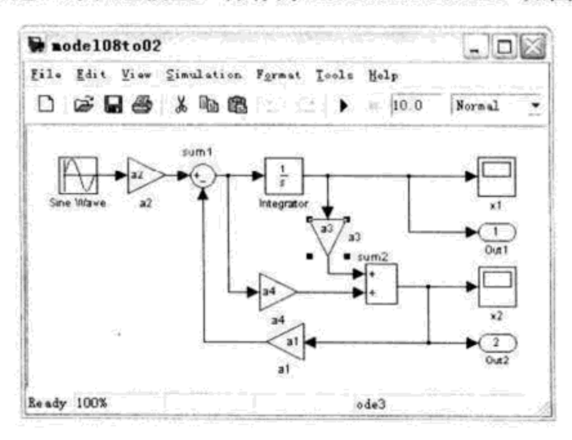


图 8.4　具有代数环的系统模型

Simulink 模型中存在一个代数环，由模块 sum1、a4、sum2 和 a1 构成一个代数环路。代数环中的每个模块都具有一个相同的特征，就是每个模块的输入和输出之间只有代数关系。这种代数关系，在时间上讲就是没有延迟，在物理模型上讲就是无惯性。

下面列举 Simulink 中这种无惯性的模块，如表 8.1。

表 8.1　构成代数环的模块类型

直通模块	模块举例
增益模块	Gain
加减运算	Add、Subtract、Sum
乘除运算	Divide、Product
数学模块	Abs、Sign、Logical
非线性模块	Saturation、Quantizer、Relay
状态方程中 D 矩阵非零	State-Space
分子分母同阶的传递函数模块	Transfer-Fcn
零极点数相等的零极点增益模块	Zero-Pole
积分模块的初值条件输入口	Integrator

(2) 对方程进行处理。将方程组第二个方程代入第一个方程，可以得到：

$$\dot{x}_1=-a_1a_3x_1-a_1a_4\dot{x}_1+a_2u \tag{1}$$

注意：

- 方程两边都出现状态导数，这就是代数环的方程表现形式之一。

- 本例所提出的方程组比较简单，所以非常容易看出代数环的问题。但对于稍微复杂的系统，就很难直接判断其是否含有代数环。

(3) 代数环的处理方法。对于上面的方程或方程组，计算机并不会自动将方程进行整理，而且把方程两边进行合并。虽然 Simulink 处理代数环的解算程序采用比较鲁棒的 Newton-Raphson 方法，却仍不能够保证结果是收敛的。

通常处理代数环的方法：

- 人工排除代数环

通过手工整理方程，得到如下方程组：

$$\dot{x}_1 = -\frac{a_1 a_3}{1+a_1 a_4} x_1 + \frac{a_2}{1+a_1 a_4} u \tag{2}$$

$$x_2 = \frac{a_3}{1+a_1 a_4} x_1 + \frac{a_2 a_4}{1+a_1 a_4} u \tag{3}$$

简化如下：

$$\dot{x}_1 = -k_1 x_1 + k_2 u \tag{4}$$

$$x_2 = k_3 x_1 + k_4 u \tag{5}$$

其中：$k_1 = \frac{a_1 a_3}{1+a_1 a_4}$，$k_2 = \frac{a_2}{1+a_1 a_4}$，$k_3 = \frac{a_3}{1+a_1 a_4}$，$k_4 = \frac{a_2 a_4}{1+a_1 a_4}$。

说明： 如果代数环对计算速度的影响不大，可以忽略；如果代数环对计算速度的影响很大，通常可以加入记忆模块或者代数约束模块，消除模块的代数环。

- 利用 Memory 模块消除代数环

(4) 建立 Simulink 模型。首先对手工整理的方程进行建模。对方程(2)和(3)建立 Simulink 模型，保存为 model08to03.mdl，如图 8.5 所示。

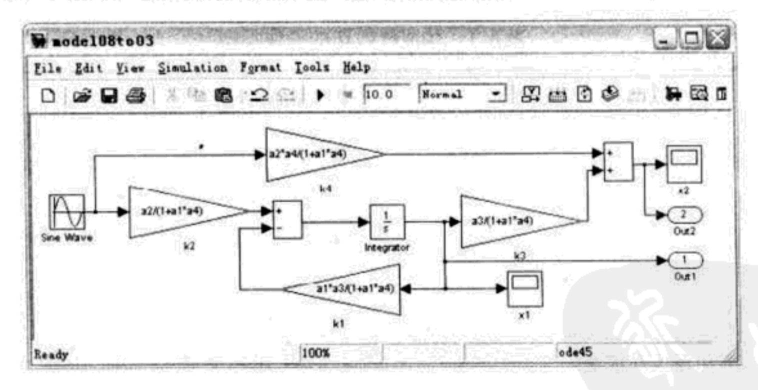


图 8.5 整理方程组消除代数环后的 Simulink 模型

利用 Memory 模块消除代数环。在图 8.4 所示 Simulink 模型的基础上，添加一个记忆环(Memory)模块，保存为 model08to04.mdl，如图 8.6 所示。

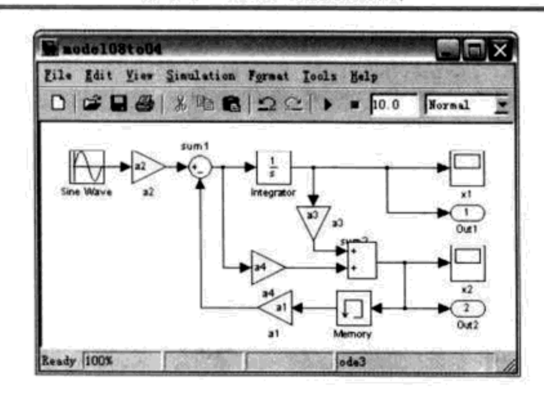


图 8.6　添加记忆模块的 Simulink 模型

(5) 运行模型。为了仿真和计算方便，编写 M 程序，保存到工作目录，并命名为 sfun08to02.m，内容如下：

```
%mfun08to02.m
%www.SciEi.com
clear
a1=1;
a2=1;
a3=0.5;
a4=1;
[t1,s,x11,x21]=sim('model08to02');%运行带有代数环的 Simulink 模型
[t2,s,x12,x22]=sim('model08to03');%运行没有代数环的 Simulink 模型
[t3,s,x13,x23]=sim('model08to04');%运行带有记忆模块的 Simulink 模型

figure(1)
%绘制输出 x1 的结果比较图
plot(t1,x11);
hold on
plot(t1,x12,'k*');
hold on
plot(t1,x13,'r:','LineWidth',2);
legend('有代数环','无代数环','记忆模块')

figure(2)
%绘制输出 x2 的结果比较图
plot(t1,x21);
hold on
plot(t1,x22,'k*');
hold on
plot(t1,x23,'r:','LineWidth',2);
legend('有代数环','无代数环','记忆模块')
```

在 MATLAB 命令窗口输入以下命令：

```
>> algebraicloop04
Found algebraic loop containing:
  'algebraicloop01/a4'
  'algebraicloop01/Sum1'
  'algebraicloop01/a1'
  'algebraicloop01/Sum' (algebraic variable)
```

运算过程中会在 MATLAB 命令窗口出现警告，还会弹出模型 algebraicloop01.mdl、algebraicloop02.mdl 和 algebraicloop03.mdl 模型的运行结果对比图，如图 8.7 和图 8.8 所示。

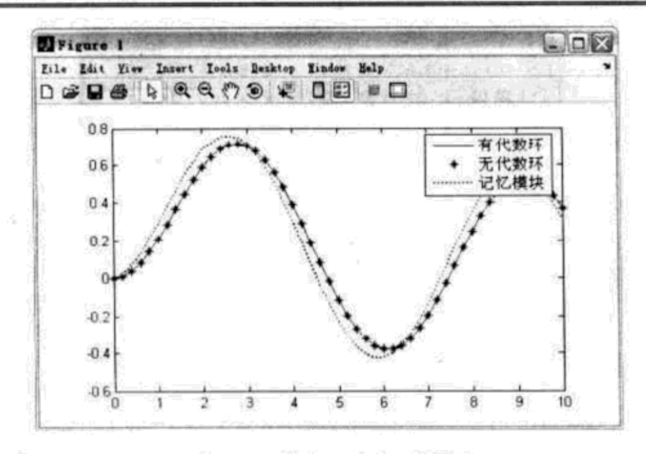


图 8.7 状态 x1 运行结果图

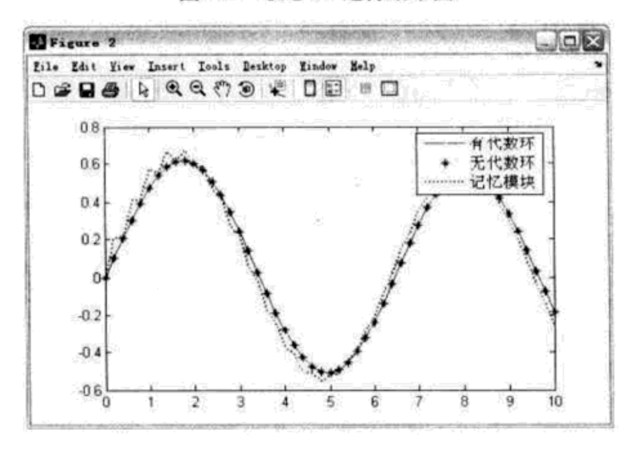


图 8.8 状态 x2 运行结果图

结果说明：

- 这 3 个模型的所有系统设置均采用默认参数，求解器均为 ode45，均为自动步长。
- 从 MATLAB 命令窗口中的运行结果可以看出，MATLAB 能够自动检测 Simulink 模型中的代数环。
- 从运算的结果可以看出带有记忆模块的模型图结果和其他的结果有一定的区别，结果精确度受到影响。
- 带有代数环的 Simulink 模型计算速度相对来说较慢。

第 9 章　连续系统、离散系统和混合系统

在 Simulink 中，可以建立 3 种系统模型，即连续系统、离散系统和混合系统。连续系统使用微分方程描述，离散系统使用差分方程描述，离散-连续混合系统采用差分-微分联立方程描述。

本章详细介绍了连续系统、离散系统和混合系统的基本模块，列举了大量的实例。

本章主要内容包括：

- 连续系统建模
- 离散系统建模
- 离散-连续混合系统建模

9.1　连续系统建模

9.1.1　线性系统

连续系统通常都是用微分方程描述的系统，而现实世界中的多数实际系统也都是连续变化的。利用 Simulink 模型建模时，通常使用 Continuous 模块库，Math operations 模块库和 Nonlinear 模块库中的模块。在此就不详细介绍每一个模块的使用方法了，具体可以参看前面的章节。

由于连续系统分为线性系统和非线性系统。虽然非线性系统是绝对的，但是不利于系统的分析和设计，通常的处理方法是将非线性近似化为线性系统，所以对线性系统的了解是非常有必要的。

要对线性系统建模，通常都要使用到积分模块。

1. 积分模块的使用

【例 0901】直接使用积分模块默认设置。

操作步骤如下：

(1) 构造 Simulink 模型如图 9.1 所示，保存文件为 model09to01.mdl，所有模块均采用默认设置。

(2) 运行仿真，在 Simulink 窗口中选择 Simulation | Start 命令，进行系统仿真。

(3) 查看结果，如图 9.2 所示。

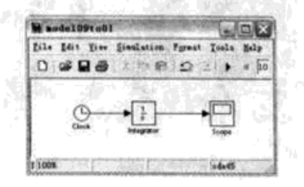


图 9.1　model09to01.mdl 模型图

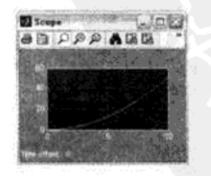


图 9.2　仿真结果

【例 0902】利用上升信号进行复位积分。

操作步骤如下：

(1) 构造 Simulink 模型如图 9.3 所示，保存文件为 model09to02.mdl。

(2) 双击积分模块，在弹出的对话框中在 External reset 下拉列表框中选择 rising 选项，在 Initial condition source 下拉列表框中选择 internal 选项，在 Initial condition 文本框中输入 5，单击 OK 按钮完成设置并关闭参数对话框。

(3) 运行仿真，在 Simulink 窗口中选择 Simulation | Start 命令，进行系统仿真。

(4) 查看结果，如图 9.4 所示。

说明： 由于模块采用上升信号进行复位积分设置，当控制信号过零时，系统回到初始值，继续进行积分运算。

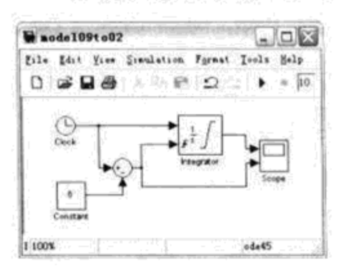


图 9.3　利用上升信号进行复位积分操作模型

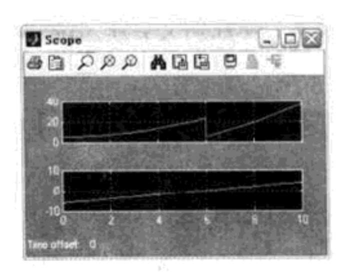


图 9.4　结果图

【例 0903】利用下降信号进行复位积分。

操作步骤如下：

(1) 构造 Simulink 模型如图 9.5 所示，保存文件为 model09to03.mdl。

(2) 双击积分模块，在弹出的对话框中在 External reset 下拉列表框中选择 falling 选项，在 Initial condition source 下拉列表框中选择 internal 选项，在 Initial condition 文本框中输入 5，单击 OK 按钮完成设置并关闭参数对话框。

(3) 运行仿真，在 Simulink 窗口中选择 Simulation | Start 命令，进行系统仿真。

(4) 查看结果，如图 9.6 所示。

说明： 由于模块采用下降信号进行复位积分设置，当控制信号过零时，系统回到初始值，继续进行积分运算。

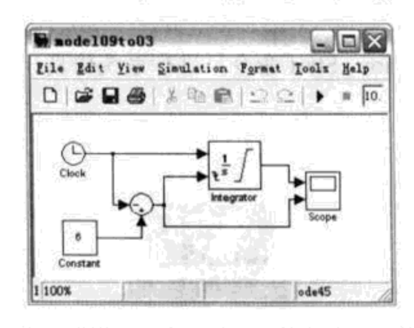


图 9.5　利用下降信号进行复位积分操作模型

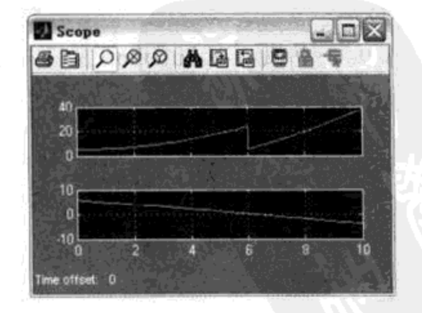


图 9.6　结果图

【例 0904】利用双向信号进行复位积分

操作步骤如下：

(1) 构造 Simulink 模型如图 9.7 所示，保存文件为 model09to04.mdl。

- 双击积分模块，在弹出的对话框中在 External reset 下拉列表框中选择 rising 选项，在 Initial condition source 下拉列表框中选择 internal 选项，在 Initial condition 文本框中输入 5，单击 OK 按钮完成设置并关闭参数对话框。
- 双击 Sine Wave 模块，其中 Amplitude 为 10，Frequency 为 3，单击 OK 按钮完成设置并关闭参数对话框。

(2) 运行仿真，在 Simulink 窗口中选择 Simulation | Start 命令，进行系统仿真。

(3) 查看结果，如图 9.8 所示。

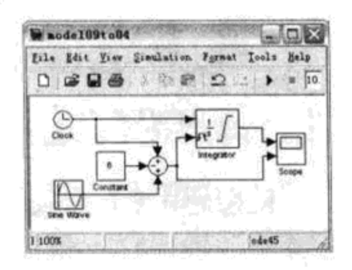


图 9.7　利用双向信号进行复位积分操作模型

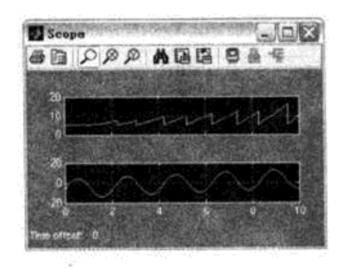


图 9.8　结果图

说明： 由于模块采用双向号进行复位积分设置，当控制信号过零时，系统回到初始值，继续进行积分运算。

【例 0905】利用水平信号进行复位积分。

操作步骤如下：

(1) 构造 Simulink 模型如图 9.9 所示，保存文件为 model09to05.mdl。

- 双击积分模块，在弹出的对话框中在 External reset 下拉列表框中选择 level 选项，在 Initial condition source 下拉列表框中选择 internal 选项，在 Initial condition 文本框中输入 5，单击 OK 按钮完成设置并关闭参数对话框。
- 双击 Pulse Generator 模块，在弹出的对话框中设置 Amplitude 为 6，Period 为 3，其他采用默认值，单击 OK 按钮完成设置并关闭参数对话框。

(2) 运行仿真，在 Simulink 窗口中选择 Simulation | Start 命令，进行系统仿真。

(3) 查看结果，如图 9.10 所示。

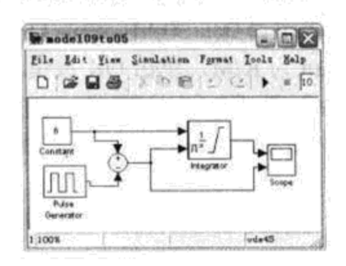


图 9.9　利用水平信号进行复位积分操作模型

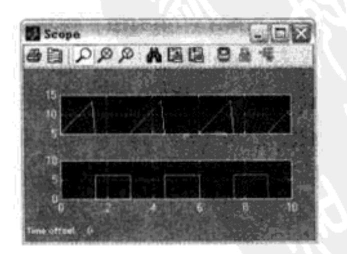


图 9.10　结果图

【例 0906】利用阶跃信号进行复位积分

操作步骤如下：

(1) 构造 Simulink 模型如图 9.11 所示，保存文件为 lmodel09to06.mdl。

(2) 双击积分模块，在弹出对话框中在 External reset 下拉列表框中选择 rising 选项，在 Initial condition source 下拉列表框中选择 internal 选项，在 Initial condition 文本框中输入 5，单击 OK 按钮完成设置并关闭参数对话框。

(3) 运行仿真，在 Simulink 窗口中选择 Simulation | Start 命令，进行系统仿真。

(4) 查看结果，如图 9.12 所示。

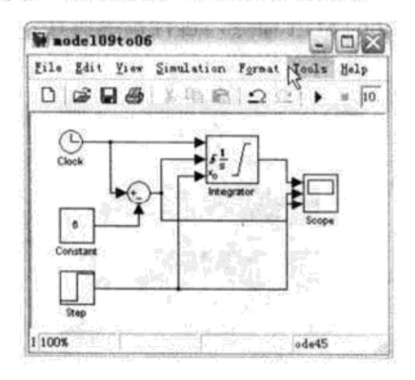


图 9.11 利用阶跃信号进行复位积分操作模型

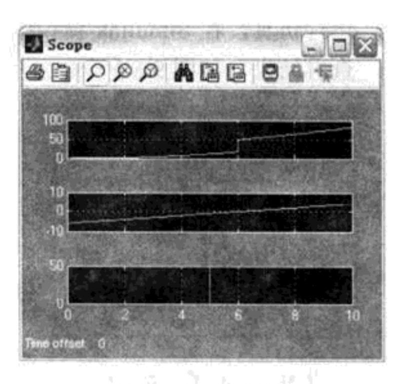


图 9.12 结果图

2. 通过积分模型构造动力学方程

实际系统可以抽象为初始状态为 0 的二阶微分方程或方程组，下面建立一个单层楼房的受到地震影响的动力学模型。

【例 0907】一个单层楼房，结构地震输入为美国 EI-Centro 南北方向的地震波，延续时间长 20s，结构质量为 M=2923.38kg，刚度为 K=1391.06kN/m，阻尼为 C=6373.74Ns/m，求结构在地震作用下的地震响应。

解题步骤如下：

(1) 建立楼房结构动力学方程：$M\ddot{x}(t)+C\dot{x}(t)+Kx(t)=f(t)$。

(2) 变换动力学方程：$\ddot{x}=\frac{f(t)}{M}-\frac{C}{M}\dot{x}-\frac{K}{M}x$。

(3) 构造 Simulink 模型。建立 Simulink 模型如图 9.13 所示，保存文件名为 model09to07.mdl。然后设置模块。

- 双击 From Workspace 模块，在弹出参数对话框中设置 Data 为 EI，单击 OK 按钮。
- 双击 Gain 模块，在弹出参数对话框中设置 Gain 为 K/M，单击 OK 按钮。
- 双击 Gain1 模块，在弹出参数对话框中设置 Gain 为 C/M，单击 OK 按钮。
- 双击 To Workspace 模块，在弹出对话框中设置 Variable name 为 x，单击 OK 按钮。
- 积分模块保持为默认设置不变。

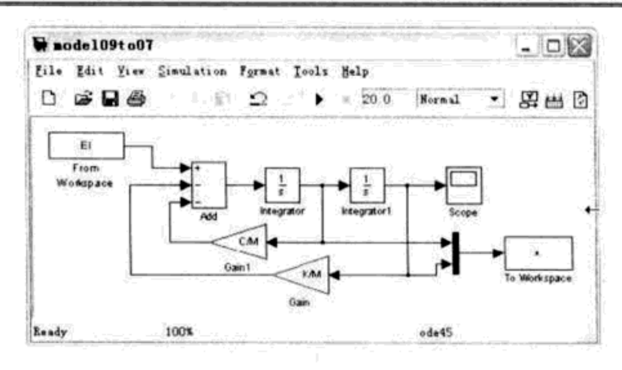


图 9.13 动力学系统 Simulink 模型

(4) 进行仿真。编写 M 文件来输入系统仿真参数，运行 Simulink 模型以及查看结果，保存文件名为 mfun09to01.m，并在 MATLAB 命令窗口中输入：

```
>> mfun09to01
```

从中可以看出通过 M 文件调用 Simulink 模型的好处，对于一个参数经常修改的模型，可以通过编写一个循环来实现这个目的。在运行 mfun09to01.m 文件之前，必须将数据文件 EI.mat 复制到同一目录下。

mfun09to01.m 内容文件如下：

```
% mfun09to01.m
% copyright@www.SciEi.com
load('EI.mat')                          %调入EI地震数据
M=2923.38;                              %设置系统质量
K=1391060;                              %设置系统刚度
C=6373.74;                              %设置系统阻尼
sim('model09to07');                     %进行系统仿真
figure(1)
plot(EI(:,1),EI(:,2));                  %绘制地震加速度
set(gca,'FontSize',12)
xlabel('Time');ylabel('Acceleration of Gravity')
grid on
figure(2)
plot(tout,x.signals.values(:,1));       %绘制速度结果
set(gca,'FontSize',12)
xlabel('Time');ylabel('Velocity')
grid on
figure(3)
plot(tout,x.signals.values(:,2));       %绘制位移结果
set(gca,'FontSize',12)
xlabel('Time');ylabel('Displacement')
grid on
```

(5) 仿真结果。运行上述程序之后，会得到如图 9.14～图 9.16 所示的结果。图 9.14 为地震加速度曲线，也就系统的外部输入。图 9.15 为结构响应速度曲线，图 9.16 为结构响应位移曲线。

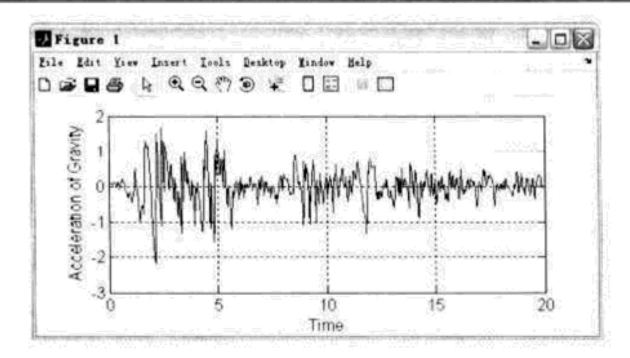


图 9.14　EI 波地震加速度

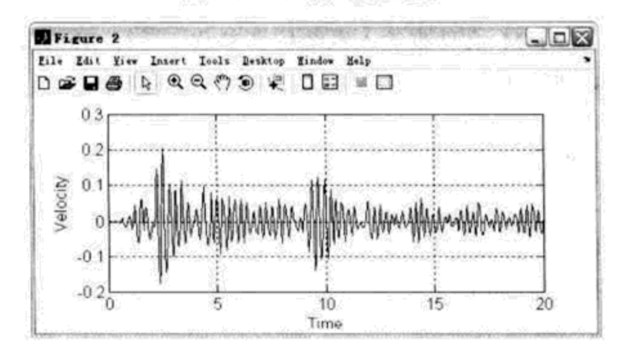


图 9.15　结构响应速度曲线

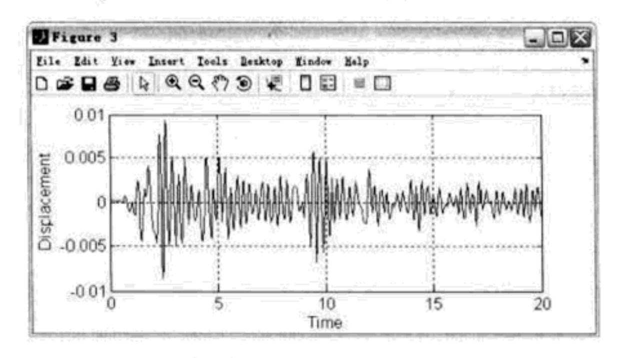


图 9.16　结构响应位移曲线

3. 通过传递函数建立系统模型

【例 0908】依照【例 0907】中的参数，建立系统模型的传递函数。

操作步骤如下：

(1)　激励动力学方程：$M\ddot{x}(t)+C\dot{x}(t)+Kx(t)=f(t)$。

(2)　对方程进行 Laplace 变换：$s^2X(s)+CsX(s)+KX(s)=F(s)$。

整理得到：$G(s)=\dfrac{X(s)}{F(s)}=\dfrac{1}{s^2+Cs+K}$。

(3)　建立 Simulink 模型。Simulink 模型如图 9.17 所示，保存为 model09to08.mdl。

- 双击 From Workspace 模块，在弹出参数对话框中设置 Data 为 EI，单击 OK 按钮。
- 双击 Transfer Fcn 模块，在弹出参数对话框中，设置 Numerator 为[1]，Denominator 为[1 C/M K/M]，如图 9. 18 所示，单击 OK 按钮。
- 双击 To Workspace 模块，在弹出对话框中设置 Variable name 为 x，单击 OK 按钮。
- 微分模块保持为默认设置不变。

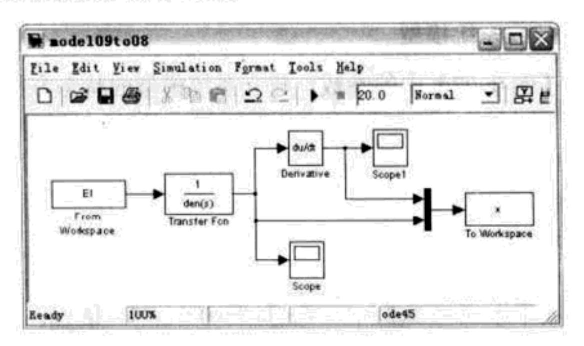


图 9. 17　传递函数 Simulink 模型

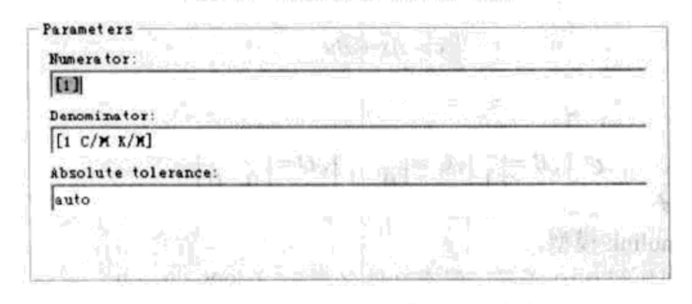


图 9. 18　Transfer Fcn 模块参数设置

(4) 进行仿真。编写 M 文件来输入系统仿真参数，运行 Simulink 模型以及查看结果，保存文件名为 mfun09to02.m。

在 MATLAB 命令窗口输入：

```
>> mfun09to02
```

mfun09to02.m 文件如下：

```
% mfun09to02.m
% copyright@www.SciEi.com
load('EI.mat')%调入 EI 地震数据
M=2923.38;%设置系统质量
K=1391060;%设置系统刚度
C=6373.74;%设置系统阻尼
sim('model09to08');%进行系统仿真
figure(1)
plot(tout,x.signals.values(:,1));%绘制速度结果
set(gca,'FontSize',12)
xlabel('Time');
ylabel('Velocity')
grid on
figure(2)
plot(tout,x.signals.values(:,2));%绘制位移结果
```



```
set(gca,'FontSize',12)
xlabel('Time');
ylabel('Displacement')
grid on
```

(5) 仿真结果。在 MATLAB 命令窗口中运行上述程序之后，会得到图 9.15 和图 9.16 所示的结果。图 9.15 为结构速度响应曲线，图 9.16 为结构位移响应曲线。

4. 通过状态方程建立系统模型

【例 0909】依照【例 0907】中的参数，建立系统模型的状态方程。

操作步骤如下：

(1) 激励动力学方程：

$M\ddot{x}(t)+C\dot{x}(t)+Kx(t)=f(t)$

(2) 对方程进行状态方程变换。

令 $x(1)=x$，$x(2)=\dot{x}$，那么方程可变换为：

$$x=\begin{bmatrix}\dot{x}(1)\\ \dot{x}(2)\end{bmatrix}=\begin{bmatrix}x(2)\\ \frac{f}{M}-\frac{C}{M}x(2)-\frac{K}{M}x(1)\end{bmatrix}=\begin{bmatrix}0 & 1\\ -\frac{K}{M} & -\frac{C}{M}\end{bmatrix}\begin{bmatrix}x(1)\\ x(2)\end{bmatrix}+\begin{bmatrix}0\\ 1\end{bmatrix}\frac{f}{M}$$

整理得

$$\begin{cases}\dot{x}=Ax+Bu\\ y=Cx+Du\end{cases}$$

其中 $A=\begin{bmatrix}0 & 1\\ -\frac{K}{M} & -\frac{C}{M}\end{bmatrix}$，$B=\begin{bmatrix}0\\ 1\end{bmatrix}$，$C=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$，$D=\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$。

(3) 建立 Simulink 模型。

Simulink 模型如图 9.19 所示，保存文件名为 model09to09.mdl：

- 双击 From Workspace 模块，在弹出的参数对话框中设置 Data 为 EI，单击 OK 按钮。
- 双击 State-Space 模块，在弹出的参数对话框中，设置 A 为 A，设置 B 为 B，设置 C 为 C，设置 D 为 D，如图 9.20 所示，单击 OK 按钮。
- 双击 To Workspace 模块，在弹出的对话框中设置 Variable name 为 x，单击 OK 按钮。

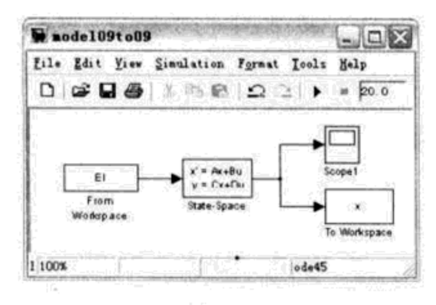


图 9.19 状态方程 Simulink 模型



图 9.20 参数设置对话框

(4) 进行仿真。编写 M 文件来输入系统仿真参数，运行 Simulink 模型以及查看结果，保存文件名为 mfun09to03.m。

在 MATLAB 命令窗口输入：

```
>> mfun09to03
```

mfun09to03.m 文件如下：

```
% mfun09to03.m
% copyright@www.SciEi.com
load('EI.mat')%调入 EI 地震数据
M=2923.38;%设置系统质量
K=1391060;%设置系统刚度
C=6373.74;%设置系统阻尼
A=[0 1;-K/M -C/M];
B=[0;1];
C=eye(2);
D=zeros(2,1);
sim('model09to09');%进行系统仿真
figure(1)
plot(tout,x.signals.values(:,1));%绘制速度结果
set(gca,'FontSize',12)
xlabel('Time');
ylabel('Velocity')
grid on
figure(2)
plot(tout,x.signals.values(:,2));%绘制位移结果
set(gca,'FontSize',12)
xlabel('Time');
ylabel('Displacement')
grid on
```

(5) 仿真结果。运行上述程序之后，会得到图 9.15 和图 9.16 所示结果。

9.1.2　非线性系统

线性是相对的，非线性是绝对的。光靠 Simulink 中的线性模块是不能够完成所有任务的，所以 Simulink 还提供了大量的非线性模块，如继电器模块 Relay，死区模块 Dead zone，饱和模块 Saturation 等。方法同线性系统类似，只要将这些非线性模块添加到模型中适当的位置即可。

9.2　离散系统建模

离散系统通常都是用差分方程来描述的系统，而实验中，都是采用离散采样。利用 Simulink 模型建模时，通常使用 Discret 模块库，Math operations 模块库和 Sink 模块库和 Source 模块库中的模块。在此不详细介绍每一个模块的使用方法，具体可以参看前面的章节。

9.2.1　模块介绍

在此只介绍几个主要的模块：

- 单位延迟模块(Unit delay)

  实现计算 $y(k)=u(k-1)$，是传递函数的特殊形式，即为：$\frac{1}{z}$。

- 离散传递函数

  传递函数主要有 3 种形式，分别如下：

  - 离散传递函数模块 Discrete Transfer Fcn，函数的分子和分母都是以 z 的降幂形式升序排列的。
  - 零极点传递函数模块 Discrete Pole-Zero，函数的分子和分母都是以 z 因式分解形式表示的。
  - 离散滤波器模块 Discrete Filter，函数的分子和分母都是以 z^{-1} 的降幂形式升序排列。

- 离散状态方程模块(Discrete State-space)

  离散模块的具体形式如下：

$$\begin{cases} y(n) = Cx(n) + Du(n) \\ x(n+1) = Ax(n) + Bx(n) \end{cases}$$

  确定矩阵 A、B、C、D，就确定离散系统，矩阵还可通过连续系统转换 c2d 函数得到。

- 零阶保持器(Zero-Order hold)

  输入端是采样器，输出端是常数保持器，实现 $y(kT) = u(t)$。

9.2.2 离散系统实例

1. 单速率离散系统

【例 0910】依照【例 0907】中的参数，建立系统模型的状态方程，确定连续系统矩阵 A、B、C、D，然后通过转换函数，得到离散系统模型的状态方程的矩阵 A、B、C、D。

建立系统模型的状态方程的操作步骤如下：

(1) 获取离散系统模型的状态方程的矩阵 A、B、C、D，利用函数 c2d。

sys=ss(A,B,C,D);%建立连续状态方程

sysd=c2d(sys,0.02,'zoh')%得到离散状态方程

(2) 首先建立 Simulink 模型。建立的模型如图 9.21 所示，保存文件名为 model09to10.mdl。

- 双击 From Workspace 模块，在弹出参数对话框中设置 Data 为 EI，单击 OK 按钮。
- 双击 Discrete State-Space 模块，在弹出对话框中设置：A 为 sysd.a，B 为 sysd.b，C 为 sysd.c，D 为 sysd.d，Initial conditions 为 0，Sample time 为 0.02。如图 9.22 所示，单击 OK 按钮。
- 双击 To Workspace 模块，在弹出对话框中设置 Variable name 为 x，单击 OK 按钮。

(3) 运行仿真。编写 M 文件来输入系统仿真参数，运行 Simulink 模型以及查看结果，保存文件名为 mfun09to04.m。

在 MATLAB 命令窗口输入：

```
>> mfun09to04
```

mfun09to04.m 文件如下：

```
% mfun09to04.m
% copyright@www.SciEi.com
load('EI.mat')%调入EI地震数据
M=2923.38;%设置系统质量
K=1391060;%设置系统刚度
C=6373.74;%设置系统阻尼
A=[0 1;-K/M -C/M];B=[0;1];C=eye(2);D=zeros(2,1);
sys=ss(A,B,C,D);%建立连续状态方程
sysd=c2d(sys,0.02,'zoh')%得到离散状态方程
sim('model09to10');%进行系统仿真
figure(1)
plot(tout,x.signals.values(2:end,1));%绘制速度结果
set(gca,'FontSize',12)
xlabel('Time');ylabel('Displacement');title('离散系统')
grid on
figure(2)
plot(tout,x.signals.values(2:end,2));%绘制位移结果
set(gca,'FontSize',12)
xlabel('Time');ylabel('Velocity');title('离散系统')
grid on
```

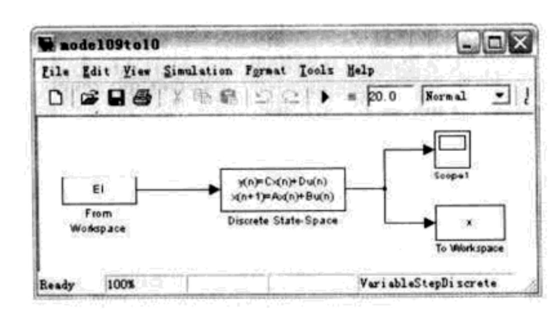


图 9.21　离散系统 Simulink 模型

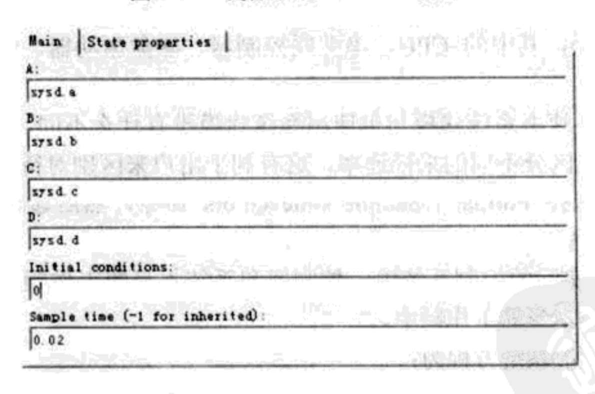


图 9.22　Discrete State-Space 模块参数设置

(4) 仿真结果。运行上述程序之后，会得到图 9.23 和图 9.24 所示结果。图 9.23 为结构响应速度曲线，图 9.24 为结构响应位移曲线。可比较结果图 9.23 与图 9.15，以及图 9.24 与图 9.16，可以发现，两种方法得到近乎一样的结果。

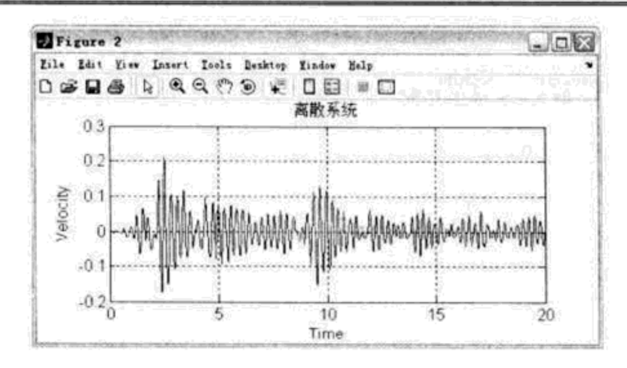


图 9.23 离散系统仿真速度输出结果

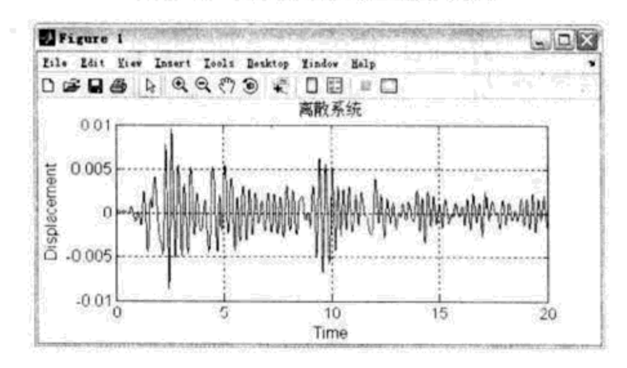


图 9.24 离散系统仿真位移输出结果

2. 多速率离散系统

在离散系统中往往会遇到同一个系统具有多个不同采样速率，不同时间偏移的子系统。例如计算机系统，其中的 CPU、串并联控制器、磁盘驱动器、输入键盘就采用不同的工作速率。

从原理上讲，多速率系统建模与单速率系统建模没有什么不同。在 Simulink 中，可以通过不同的颜色来区分不同的采样速率，这有利于用户来区别对待。这个功能可以通过菜单项进行设置，选择 Format | Sample time colors 命令，然后通过选择 Edit | Update Diagram 命令来更新窗口。

【例 0911】在实际的控制系统中，控制器和系统本身的工作频率是不一样，控制器频率往往会低于系统本身的工作频率。

假设某系统的离散系统方程为：

$$\begin{cases} x_1(k+1) = -0.5x_1(k) + 0.4\sin x_2(k) - u_1(k) \\ x_2(k+1) = 0.5x_1(k) - 0.035x_2(k) - u_2(k) \end{cases}$$

式中 $u(k)$ 是系统输入。该过程采样周期为 0.05 秒，控制器采样周期为 0.1 秒的比率控制器，显示系统的更新周期为 0.2 秒。

(1) 建立 Simulink 模型。建立的模型如图 9.25 所示，保存文件名为 model09to11.mdl。

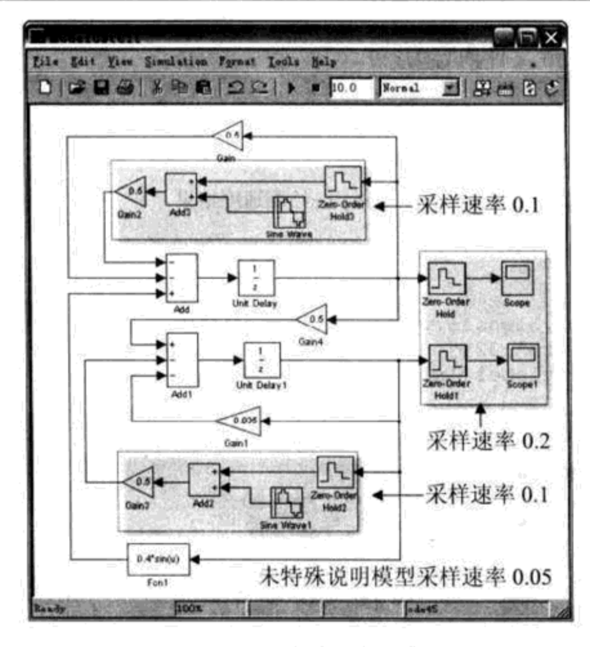


图 9.25　Simulink 模型

- 模块 Zero-Order Hold 和模块 Zero-Order Hold1 采样为 0.2。
- 模块 Zero-Order Hold2 和模块 Zero-Order Hold3 采样为 0.1。
- 模块 Add2、模块 Add3、模块 Gain2 和模块 Gain3 的 Sample time 为 0.1，其他保持默认设置。
- Sine Wave1 参数对话框中设置 Amplitude 为 2，Sample time 为 0.1。
- Sine Wave 参数对话框中设置 Amplitude 为 1，Sample time 为 0.1。
- 双击 Scope 模块，在弹出参数对话框中单击按钮，在弹出对话框中单击 Data History 选项卡，选中 Save data to workspace 复选框，在 Varaiable name 文本框中输入 x1，如图 9.26 所示。按同样的方法设置 Scope1 模块，在 Varaiable name 文本框中输入 x2。



图 9.26　Scope 模块参数设置对话框

(2) 运行仿真。在 MATLAB 命令窗口输入如下命令：

```
>> mfun09to05
```

其中 mfun09to05.m 文件内容为：

```
% mfun09to05.m
% copyright@www.SciEi.com
sim('model09to11');%进行系统仿真
figure(1)
plot(x1.time,x1.signals.values);%绘制速度结果
set(gca,'FontSize',12)
xlabel('Time');ylabel('x1');title('离散系统')
grid on
figure(2)
plot(x2.time,x2.signals.values);%绘制位移结果
set(gca,'FontSize',12)
xlabel('Time');ylabel('x2');title('离散系统')
```

(3) 运行结果。运行结果之后，会弹出两个结果图如图 9.27 和图 9.28 所示。

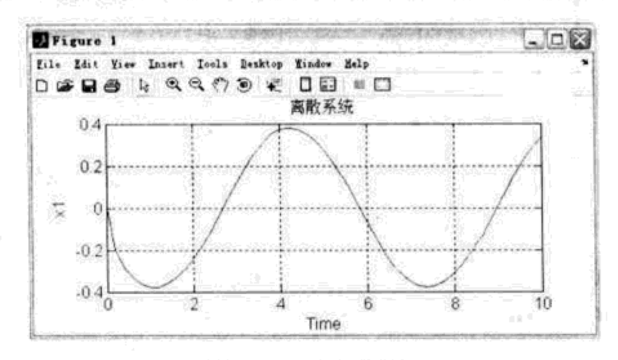


图 9.27 x1 结果曲线

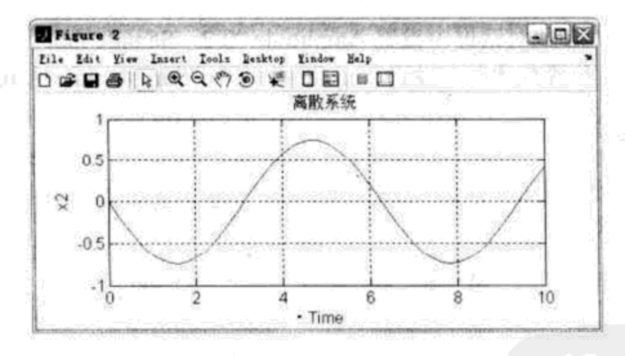


图 9.28 x2 结果曲线

9.3 离散-连续混合系统建模

在现代控制系统中，被控系统均为连续的，如建筑结构主动控制中，受控系统建筑结构为一个连续系统，而控制系统为离散子系统。对于这类离散-连续混合系统，模型参数设置中的所有求解器都可以用。

【例 0912】对一个三层建筑结构，如图 9.29 所示，利用连续系统进行建模，利用离散控制系统进行控制。

操作步骤如下：

(1) 对系统进行动力学建模。

考虑地震和控制律共同作用下 n 个自由度结构的运动方程：

$$M\ddot{X}(t)+C\dot{X}(t)+KX(t)=Bu(t)+MI\ddot{g}(t) \tag{1}$$

式中 M、C 和 K 分别为 $n\times n$ 阶的质量矩阵、阻尼矩阵和刚度矩阵；B 为 $n\times r$ 阶控制律位置矩阵；X 为 $n\times 1$ 阶的结构位置向量；u 为 $r\times 1$ 阶控制律向量；$\ddot{g}$ 为地震加速度；I 为 n 维数的向量。

$$M=2500*\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix},\quad K=1.25*10^6*\begin{bmatrix}2&-1&0\\-1&2&-1\\0&-1&2\end{bmatrix},\quad C=5000*\begin{bmatrix}2&-1&0\\-1&2&-1\\0&-1&2\end{bmatrix}$$

(2) 运动方程转移到状态空间方程为：

$$\dot{Z}(t)=AZ(t)+DU(t)+F(t) \tag{2}$$

式中：$Z(t)=\begin{bmatrix}X(t)\\\dot{X}(t)\end{bmatrix}$；$A=\begin{bmatrix}0&I\\-M^{-1}K&-M^{-1}C\end{bmatrix}$；$D=\begin{bmatrix}0\\M^{-1}B\end{bmatrix}$；$F(t)=\begin{bmatrix}0\\I\ddot{g}(t)\end{bmatrix}$。

(3) 建立 Simulink 模型。本例并不注重控制方法和控制的效果，而是为了演示离散-连续混合系统，采用简单的 PID 离散控制系统。

建立模型如图 9.30 所示 Simulink 模型，保存为 model09to12.mdl，模型参数设置如下：

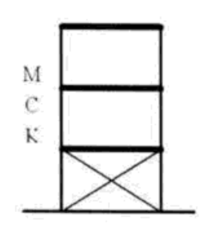


图 9.29　建筑建构模型

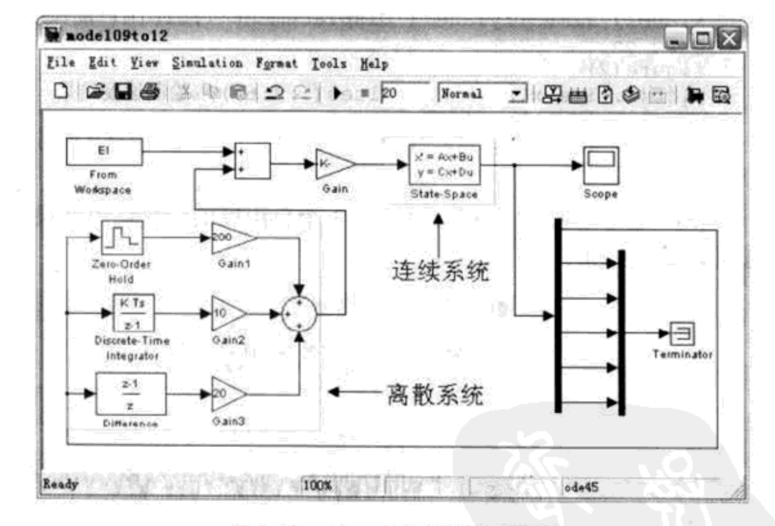


图 9.30　Simulink 控制系统模型

- Gain 模块中的参数 Gain 设置为[0 0 1]'。
- Gain1 模块中的参数 Gain 设置为 200。
- Gain2 模块中的参数 Gain3 设置为 10。
- Gain3 模块中的参数 Gain 设置为 20。
- 模块 State-Space 模块设置：A 为 A，B 为 B，C 为 C，D 为 D。
- 双击 From Workspace 模块，在弹出参数对话框中，设置 Data 为 EI。

- Zero-Hold 模块和 Discrete-Time Integrator 模块中的 Sample time 为 0.2。
- 双击 Scope 模块，在弹出参数对话框中单击按钮，在弹出对话框中单击 Data History 选项卡，选中 Save data to workspace 复选框，在 Variable name 文本框中输入 x。

(4) 运行仿真。在 MATLAB 命令窗口输入如下命令：

```
>> mfun09to06
```

其中 mfun09to06.m 文件内容为：

```
% mfun09to06.m
% copyright@www.SciEi.com
% 这是一个多自由度受到地震荷载的情况
clear
load('EI.mat');     % 导入地震波
M=[2500 0 0;0 2500 0;0 0 2500]; % 结构质量矩阵
K=[2500 -1250 0;-1250 2500 -1250;0 -1250 1250]*1000; % 结构刚度矩阵
C=[10000 -5000 0;-5000 10000 -5000;0 -5000 5000]; % 结构阻尼矩阵
ling=zeros(3); % 定义零矩阵
danwei=eye(3); % 定义单位矩阵
% 转换到状态空间
A=[ling danwei;-K*inv(M) -
C*inv(M)];B=[ling;danwei];C=[eye(6)];D=[ling;ling];
sim('model09to12');
figure(1)
plot(x.time,x.signals.values(:,1:3));%绘制速度结果
set(gca,'FontSize',12)
xlabel('Time');ylabel('Displacement');title('混合系统');
grid on
figure(2)
plot(x.time,x.signals.values(:,4:6));%绘制速度结果
set(gca,'FontSize',12)
xlabel('Time');ylabel('Velocity');title('混合系统');
grid on
```

(5) 运行结果。图 9.31 为结构 3 个自由度的位移输出结果，图 9.32 为结构 3 个自由度速度输出结果。

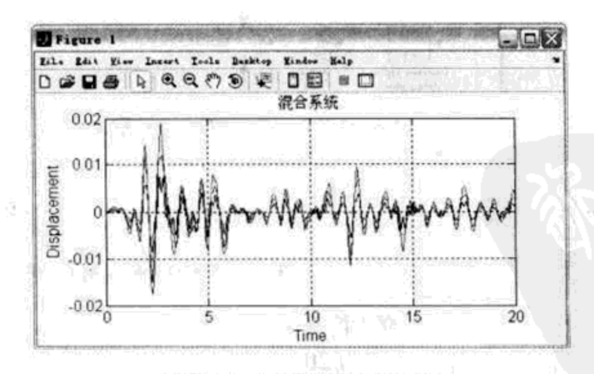


图 9.31　结构位移时程曲线

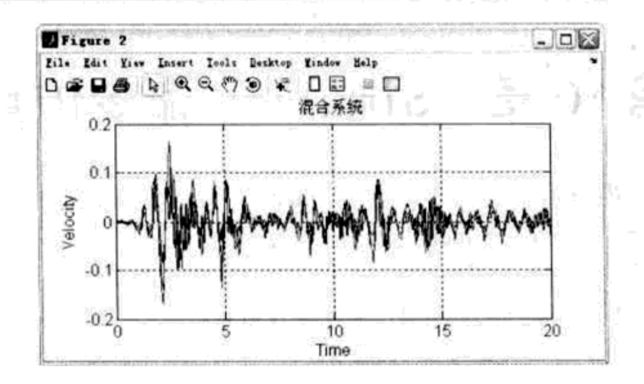


图 9.32 结构速度时程曲线

第 10 章 Simulink 命令仿真

本章将详细介绍如何通过命令行来建立 Simulink 模型，通过命令进行 Simulink 仿真和分析。介绍如何将命令行和 Simulink 模型仿真相结合。

在此简单介绍命令行仿真相较 Simulink 模型仿真有如下优点：

- 可以自动进行批处理运行仿真，重复运行仿真。
- 在仿真过程中实现动态调用外部参数。
- 对不同的输入信号下的系统相进行分析。
- 在 MATLAB 下进行 Simulink 仿真。
- 进行快速仿真。

本章主要内容包括：

- 使用命令方式建立系统模型
- 用 MATLAB 命令运行 Simulink 模型
- 非线性模型的线性化

10.1 使用命令方式建立系统模型

在前面及以后其他各章使用的建模方法、系统的仿真与分析，都是建立在 Simulink 图形模型的基础上。Simulink 的图形建模分析方法的友好界面以及强大的功能，使得用户建模更加直观方便，大大提高了仿真效率。使用 Simulink 的图形化仿真技术，能够解决通常所能够遇到的问题，但是在一些特殊的情况下，Simulink 的图形界面使得用户不能够对系统模型进行深入的操作和对系统的仿真加以修改。例如，当在仿真过程中，系统模型的参数需要进行修改，交互式输入参数，如果通过手动的修改必将不能满足各种仿真的要求。

Simulink 仿真命令比较多，现将常用建立模型的命令及其功能列于表 10.1。

表 10.1 建立模型的常用命令

命 令	功 能
new_system	创建一个新的 Simulink 空白系统模型
open_system	打开 Simulink 模型或者模块对话框
close_system	关闭 Simulink 模型或者模块对话框
save_system	保存 Simulink 模型
find_system	查找 Simulink 系统模型、模块、连线及注释
add_block	在系统模型中加入指定模块
delete_block	删除系统模型中指定模块
replace_block	替代系统模型中指定模块

续表

命　令	功　能
add_line	在系统模型中加入指定连线
delete_line	删除系统模型中指定连线
get_param	获取系统模型中的参数
set_param	设置系统模型中的参数
gcb	获得当前模块的路径名
gcs	获得当前系统模型的路径名
gcbh	获得当前模块的操作句柄
bdroot	获得最上层系统模型的名称
simulink	打开 Simulink 的模块库浏览器

下面来详细介绍各个命令的语法和详细用法：

1. new_system

1)　语法与功能

- new_system('sys')以指定的文件名创建一个新的空白的 Simulink 模型文件。
- new_system('SYS','Library')创建一个空白模块库，在 Simulink 子系统封装中介绍过模块库的概念。
- new_system('SYS','Model')创建一个空白的模型系统。
- new_system('SYS','Model','FULL_PATH_TO_SUBSYSTEM')创建一个空白系统，并引用一个子系统模型。使用此命令时，第二个参数一定要是 MODEL，第三个参数是子系统的路径。

2)　实例

- 创建名为 NewModel 的 Simulink 模型，但并不打开模型。

```
new_system('NewModel')
```

- 创建名为 mysys 的 Simulink 模型，并引用一个子系统模型，模型中调用'f14'演示模型中名为'Controller'子系统中的模块。

```
new_system('mysys','Model','f14/Controller')
```

2. open_system

1)　语法与功能

- open_system('SYS')打开一个 Simulink 系统或者子系统模型。
- open_system('BLK') 其中'BLK'是完整模块路径，打开制定模块的参数对话框。
- open_system 还可以有其他附加参数，这些参数见表 10.2。

表 10.2　open_system 附加参数及其功能

附加参数	功　能
DESTSYS	在另外一个窗口中打开 SYS 模型
force	在封装形式下打开模型

续表

附加参数	功 能
parameter	打开模块的参数对话框
property	打开模块的属性对话框
mask	打开模块的封装对话框
OpenFcn	执行模块的 OpenFcn
replace	在目标窗口中打开模型文件，重新使用目标文件，并且使得目标文件模型窗口大小和被打开的模型窗口一样大
reuse	在目标窗口中打开模型文件，重新使用目标文件，并且使得目标文件模型窗口调整到适当大小

2) 实例

- 打开'f14'模型。

```
open_system('f14');
```

- 打开'f14'模型中的 Controller 子系统。

```
open_system('f14/Controller') ;
```

- 在'reuse'模式下，在'f14/Controller'窗口中打开'f14'。

```
open_system('f14','f14/Controller','reuse');
```

3. close_system

1) 语法与功能

- close_system ('SYS') 关闭指定的系统或者子系统模型。
- close_system ('SYS',SAVEFLAG) 如果 SAVEFLAG 是 1，则以当前文件名保存模型，并关闭模型窗口。如果 SAVEFLAG 是 0，则直接关闭模型窗口，而不保存文件。
- close_system ('SYS','NEWNAME') 以指定的文件名保存模型文件，并且关闭文件。
- close_system ('BLK') 如果'BLK'是模块的完整路径，则关闭模块参数对话框。

2) 实例

- 关闭当前系统。

```
close_system
```

- 关闭'vdp'系统。

```
close_system('vdp')
```

- 以'engine'文件名保存，并关闭'engine'系统。

```
close_system('engine',1)
```

- 用'myvdp'文件名保存'vdp'系统，并关闭'vdp'系统。

```
close_system('vdp','myvdp')
```

4. save_system

1) 语法与功能

- save_system('SYS') 保存当前系统模型文件。
- save_system('SYS','NEWNAME') 将当前系统以文件名'NEWNAME'保存。
- save_system('SYS','NEWNAME','BreakLinks') 将当前系统以文件名'NEWNAME'保存模型文件，并打断模型文件中的所有关联。
- save_system('SYS','NEWNAME','LINKACTION', 'VERSION') 将指定的系统以文件名'NEWNAME',保存为 MATLAB 以前的版本，其中'LINKACTION'为''或者'BreakLinks'，其中'VERSION'为 R13SP1，'R13'，'R12P1'和'R12'。

2) 实例

- 保存当前系统。

```
save_system
```

- 保存'vdp'系统。

```
save_system('vdp')
```

- 以'myvdp'文件名保存 vdp 系统。

```
save_system('vdp','myvdp')
```

- 以'myvdp'文件名保存 vdp 系统，并打断模型中的关联。

```
save_system('vdp','myvdp','BreakLinks')
```

5. find_system

1) 语法与功能

- Systems= find_system 搜索所有打开的 Simulink 模型文件，返回一个细胞数组，内容为模型文件的完整路径，并且是按系统，子系统，模块来分开的。
- Systems=find_system('PARAM_NAME1',VALUE1,'PARAM_NAME2',VALUE2,…)

利用参数对对搜索进行限定，搜索满足要求的系统或者模块。参数具体功能及其属性见表 10.3 所示，所用到的正则表达式见表 10.4 所示。

表 10.3　参数具体功能及其属性

属　性	属性值	功　能
SearchDepth	0、1、2	说明搜索的内容(0 表示搜索打开的顶层系统，1 表示搜索顶层系统的模块或子系统，2 表示搜索顶层系统和其下面的系统)，默认是搜索所有系统
LookUnderMasks	'none'	搜索跳过封装系统
	{'graphical'}	搜索那些无工作空间和对话框的封装模块，这也是默认值
	'functional'	搜索那些无对话框的封装模块
FollowLinks	'on'| {'off'}	如果是'on'，将在库模块中搜索相关链接，默认是'off'
FindAll	'on'| {'off'}	如果是'on'，则搜索的内容包括线，端口以及注释，默认是'off'

续表

属　性	属性值	功　能
CaseSensitive	'on'| {'off'}	如果是'on'，则考虑完全匹配的条件，默认是'on'
RegExp	'on'| {'off'}	如果是'on'，搜索将认为搜索表达式是正则表达式，默认是'off'

表 10.4　正则表达式

表达式	用　法
.	匹配任何字符，例如：字符串 'a.'匹配 'aa'，'ab'，'ac'等
*	匹配前面的字符，例如，'ab*'匹配'a'，　'ab'，　'abb'等表达式'.*' 任何字符串，包括空字符串
+	匹配前面一个或多个字符，例如：　'ab+'匹配 'ab'，'abb'等
^	匹配字符串第一个字符，例如：'^a.*'匹配任何以字符'a'开头的字符串
$	匹配字符串最后一个字符，例如：'.*a$'匹配任何以字符'a'结尾的字符串
\	是紧接在后面的字符作为普通字符，这个可以解决不能使用正则字符的问题例如：搜索字符串'\\'表示任何字符串可以含有字符'\'
[]	匹配[]中的任何一个字符，例如，'f[oa]r' 匹配 'for'和'far'在[]中的字符表示特殊意义连字符(-)表示在两个字符之间的字符都将被匹配，例如'[a-zA-Z1-9]'将匹配任何数字和字母上三角(^)表明这个字符将不被匹配，例如，'f[^i]r' 匹配'far'和'for'等等，但不匹配'fir'
\w	匹配 word 字符，是[a-z_A-Z0-9]的便捷方式
\d	匹配数字，是[0-9] 的便捷方式，例如，'\d+'匹配任何一个整数
\D	匹配任何非数值字符，是[^0-9]便捷方式
\s	[\t\r\n\f]的便捷方式
\S	[^ \t\r\n\f]的便捷方式
\<WORD\>	精确的匹配 WORD，其中 WORD 是用空格和其他字符隔开的字符串，例如，'\<to\>' 匹配 'to'而不是'today'

2)　实例

- 搜索打开的 Simulink 模型，返回一个细胞数组，内容为模型文件的完整路径。

```
find_system
```

- 返回打开的模块名。

```
open_bd = find_system('type', 'block_diagram')
```

- 返回所有 Goto 模块的文件名，这些模块是未锁定 clutch 系统中的模块。

```
find_system('clutch/Unlocked','SearchDepth',1,'BlockType','Goto')
```

6. add_block

1)　语法与功能

- add_block ('SRC','DEST') 将完整路径'src'下的模块复制到完整路径'dest'中，从而

创建一个模块。复制的模块参数同原模块一模一样。名称'built-in'可以作为Simulink 内建模块的源系统名称。

- add_block('SRC','DEST','PARAMETER1',VALUE1,…) 和上面一样，也是创建一个复制模块，并且指定的参数都有特定的值。

2)　实例

- 将 simulink 系统的 Sinks 子系统中的 Scope 模块复制到 engine 系统的 timing 子系统中为 Scope1 模块。

```
add_block('simulink/Sinks/Scope', 'engine/timing/Scope1')
```

- 在 F14 系统中创建一个新的子系统，名为 controller。

```
add_block('built-in/SubSystem', 'F14/controller')
```

- 将内置增益模块复制到 mymodel 系统中，命名为 Volume，并且赋值为 4。

```
add_block('built-in/Gain', 'mymodel/Volume', 'Gain', '4')
```

7. delete_block

1)　语法与功能

delete_block('blk')，　其中'blk'是完整的模块路径，将模块从系统中删除。

2)　实例

将模块 Out1 从 vdp 系统中删除。

```
delete_block('vdp/Out1')
```

8. replace_block

1)　语法与功能

- replace_block('sys', 'blk1', 'blk2')，用'blk2'代替'sys'中所有模块中的名称或是封装类型为'blk1'。如果'blk2'是 Simulink 内置模块，那么只要模块名称就行了。如果'blk2'是另外一个系统，则需要完整的路径。
- replace_block('sys', 'Parameter', 'value', …, 'blk') 代替'sys'中的特定参数为特定值的模块，可以设定任何对的参数。

2)　实例

- 将 f14 系统中的所有 Gain 模块取代为 Integrator 模块，并且保存这些被取代模块的路径。Simulink 在取代之前将在对话框中列举所有匹配的模块。

```
RepNames = replace_block('f14','Gain','Integrator')
```

- 将 clutch 系统中未锁定子系统中名为'bv'的 Gain 模块换为 Integrator 模块。在替换之前，Simulink 将在对话框中列出所有匹配的模块。

```
replace_block('clutch/Unlocked','Gain','bv','Integrator')
```

9. add_line

1)　语法与功能

add_line('sys', 'oport', 'iport')把系统指定模块的输出端口'oport'到指定模块的输入端

'iport'连起来。'oport'和'iport'是由模块名和端口标示符组成的字符串，形式为'block/port'。大多数模块的端口从上到下，从左到右由数字标示，如'Gain/1'或者'Sum/2'。Enable、Trigger、State 和 Action 的端口都由名称标示，例如，'subsystem_name/Enable'，'subsystem_name/Trigger'，'Integrator/State'，或 if_action_subsystem_name/Ifaction'。

2) 实例

mymodel 系统中将 Sine Wave 模块中的输出端和 Mux 模块第一个输入端口连起来。

```
add_line('mymodel','Sine Wave/1','Mux/1')
```

10. delete_line

1) 语法与功能

delete_line('sys', 'oport', 'iport') 删除指定模块输出端'oport'到指定模块输入端'iport'间的连线。'oport'和'iport'是由模块名和端口标示符组成的字符串，形式为'block/port'。大多数模块的端口从上到下，从左到右由数字标示，如'Gain/1'或者'Sum/2'。Enable、Trigger、State 和 Action 的端口都由名称标示，例如，'subsystem_name/Enable'，'subsystem_name/Trigger'，'Integrator/State'，或 if_action_subsystem_name/Ifaction'。

2) 实例

删除 mymodel 系统中 Sum 模块第一输出端口到 Sum 模块第二输入端口之间连线。

```
delete_line('mymodel','Sum/1','Mux/2')
```

11. get_param

1) 语法与功能

- get_param('obj', 'parameter') 其中'obj'是一个系统或者是模块的路径名，返回指定的参数值。
- get_param({ objects }, 'parameter') 接受一个完整路径的细胞数组，使得你能够获取细胞数组中完整路径的参数值。
- get_param(handle, 'parameter')返回指定目标的参数，这个目标是一个句柄。
- get_param(0, 'parameter') 返回 Simulink 当前任务的参数值，或者是模型或模块的默认值。
- get_param('obj', 'ObjectParameters') 返回描述'obj'参数的结构体，返回结构体的每一个域相对应特定的参数，并且有参数的名称。例如，域的名称就相对应目标的参数名称。每个参数域本身包括 3 个域，名称、类型和属性。例如，参数的名称("Gain")，数据的类型(字符串)，属性(只读)。
- get_param('obj', 'DialogParameters') 返回一个细胞数组，其中包含特定模块的参数对话框参数的名称。

2) 实例

- 返回 clutch 系统中子系统 Requisite Friction 的 Inertia 模块的增益参数的值。

```
get_param('clutch/Requisite Friction/Inertia','Gain')
ans =
   1/(Iv+Ie)
```

- 返回当前选择模块的名称。

```
get_param(gcb, 'Name')
```

- 返回 Sine Wave 模块的对话框参数。

```
p = get_param('untitled/Sine Wave', 'DialogParameters')
p =
    'Amplitude'
    'Frequency'
    'Phase'
    'SampleTime'
```

12. set_param

1) 语法与功能

set_param('obj', 'parameter1', value1, 'parameter2', value2, …) 其中'obj'是一个系统或者模块路径或者是 0，对特定参数设定特定值。用 0 来设定参数的默认值。

2) 实例

- 设定 vdp 系统中的 Solver 为'ode15s'，StopTime 为'3000'。

```
set_param('vdp', 'Solver', 'ode15s', 'StopTime', '3000')
```

- 设定 vdp 系统模块 Mu 增益参数为 1000。

```
set_param('vdp/Mu', 'Gain', '1000')
```

- 设定 vdp 系统 Fcn 模块的位置。

```
set_param('vdp/Fcn', 'Position', [50 100 110 120])
```

- 设定 mymodel 系统零极点模块的零点和极点的参数。

```
set_param('mymodel/Zero-Pole','Zeros','[2 4]','Poles','[1 2 3]')
```

- 设定一个封装子系统中的增益模块，变量 k 就和 Gain 参数关联起来。

```
set_param('mymodel/Subsystem', 'k', '10')
```

- 设定 mymodel 系统中 Compute 模块中的 OpenFcn 回调参数。当双击 Compute 模块时，执行'my_open_fcn'函数。

```
set_param('mymodel/Compute', 'OpenFcn', 'my_open_fcn')
```

13. gcb

1) 语法与功能

- gcb 返回当前系统当前模块的完整路径。
- gcb('sys')返回指定系统当前模块的完整路径。

以下几种情况才能够被称之为当前模块：

- 在编辑过程中，最后单击的模块就是当前模块。
- 在包含 S 函数的模块的系统仿真过程中，当前模块就是执行 MATLAB 函数的 S 函数模块。
- 在回调过程中，当前模块是被执行回调的模块。

- 在 MaskInitialization 字符串赋值过程中，当前模块就是封装被执行的模块。

14. gcs

1) 语法与功能

gcs 返回当前系统完整路径。

- 在编辑过程中，最后单击的系统或子系统就是当前系统。
- 在包含 S 函数的模块的系统仿真过程中，当前系统或子系统就是包含执行 MATLAB 函数的 S 函数模块的系统或子系统。
- 在回调过程中，当前系统是被执行回调的系统。
- 在 MaskInitialization 字符串赋值过程中，当前系统就是封装的模块被执行的系统。

15. gcbh

1) 语法与功能

gcbh 返回当前系统中当前模块的句柄。

16. bdroot

1) 语法与功能

- bdroot 直接返回顶层系统的名称。
- bdroot('obj')其中'obj'是系统或者模块的路径名，返回包含特定目标名称的顶层系统名称。

2) 实例

bdroot(gcb)返回包含当前模块的顶层系统名称。

10.2 用 MATLAB 命令运行 Simulink 模型

Simulink 仿真除了利用 Simulink 模型窗口菜单进行模型仿真之外，还可以利用 sim 命令在 MATLAB 命令窗口或者 M 文件中运行 Simulink 模型。这就更加便于对模型的分析与仿真，可以研究不同参数、输入、初始条件的影响。

sim( )函数的调用格式是：

```
[t,x,y]=sim(model,tspan,options,ut)
```

其中 model 为 Simulink 模型名，tspan 为仿真时间控制变量，它可以为[t0,tf]，就是仿真的起始时间和终止时间。tspan 是标量，则表示终止仿真时间，options 为模型控制参数，ut 为外部输入向量。该函数返回的 t 为时间列向量，x 为状态变量构成的矩阵，其中第 i 列为第 i 状态变量的时间响应。y 为输出信号构成的矩阵，每列对应一个输出信号。

【例 1001】假设已经建立一个单自由度系统 model10to01.mdl，如图 10.1 所示。

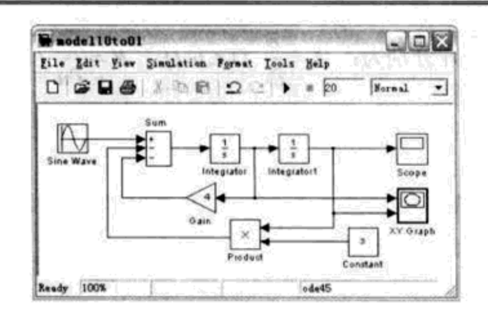


图 10.1　单自由度 Simulink 模型

在 MATLAB 命令窗口输入如下命令：

```
>> model10to01;
>> [t,x,y]=sim('simexample01',[0,20]);
```

仿真结果如图 10.2 所示。

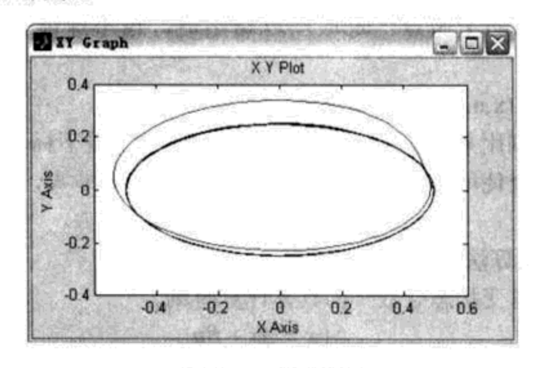


图 10.2　仿真结果

10.3　非线性模型的线性化

由于线性系统理论发展比较成熟，可以利用线性分析方法来研究非线性问题。相对于非线性系统来说，线性系统更加容易分析和设计。但是实际工程中，都是非线性的，在这种情况下，可以利用线性化理论来近似分析非线性系统。线性化实际上就是在系统工作点附近的邻域内提取系统的线性特性，再利用线性理论对非线性系统进行分析和设计。

1. 非线性系统线性化数学描述

假设非线性系统有如下形式：

$$\begin{cases}\dot{x}=f(x,u,t)\\ y=g(x,u,t)\end{cases}$$

其中：$f,\dot{x},x$ 都是 n 阶向量，u 是 m 阶向量。于是，在标称点 x、标称输入 u，和某输入时间 t 指定的情况下，可以得到以下方程：

$$\begin{cases}\delta\dot{x}=A\delta x+B\delta u\\ \delta y=C\delta x+D\delta u\end{cases}$$

其中 A、B、C、D 矩阵分别为：$A=\frac{\partial}{\partial x}f(x,u,t)$，$B=\frac{\partial}{\partial u}f(x,u,t)$，$C=\frac{\partial}{\partial x}g(x,u,t)$，$D=\frac{\partial}{\partial u}g(x,u,t)$。

说明： 标称点就是当系统状态变量导数趋于 0 时的状态变量的值。系统的工作点可以通过下面的方程来求解：

$$f(x,u,t)=0$$

Simulink 中提供了 trim()函数，可以求解系统在指定输入下的标称点，该函数为：

```
[x,u,y,dx]=trim(model,x0,u0,y0,ix,iu,iy,dx0,idx,options,t)
```

说明：

- 在所有输入输出参数中，第一个输入参数是必需的。
- 基本原理是利用数值计算机，搜索一个是状态导数的最大绝对值最小化。此命令并不保证搜索到的平衡点一定最接近初始值，也不保证平衡点存在时算法一定收敛。
- x0、u0、y0 是(x,u,y)搜索的初始值。
- ix、iu、iy 分别用来指定 x0,u0,y0 中保持不变的分量下标，是搜索受约束进行。
- dx0 与 idx 配合使用，idx 是定那些状态分量的导数非零，而 dx0 指定那些非零导数的具体值。
- options 是优化算法的参数选项设置。
- t 用来指定“时变状态导数”的具体计算时刻。

  通常简化为：
  $$\begin{cases}\dot{x}=Ax+Bu\\ y=Cx+Du\end{cases}$$

说明：

- 线性化后的系统只是在标称点、标称输入、标称时间附近的很小范围内成立。
- 这种线性化方法，使用是有局限性的，有些非线性系统会发生本质的变化。

【例 1002】考虑如图 10.3 所示非线性系统，可以看出，该系统有两个非线性环节，用 Simulink 模型可以非常容易的建立非线性模型。

说明：

- 所建立的模型中分别用输入和输出端子来表示原系统的输入和输出。
- 其中模块 Saturation 设置：Upper limit：0.5，Lower limit：-0.5，如图 10.4 所示。
- 其中模块 Transfer Fcn 设置：Numerator：[1 2],Denominator:[1 5 5]，如图 10.5 所示。
- 其中模块 Fcn 设置：Expression：u+u^3/6，如图 10.6 所示。

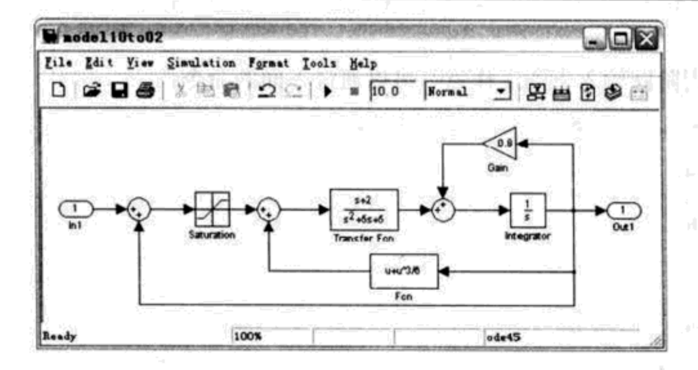


图 10.3 非线性系统

Parameters
Upper limit:
0.5
Lower limit:
-0.5
☑ Treat as gain when linearizing
☐ Enable zero crossing detection
Sample time (-1 for inherited):
-1

图 10.4 模块 Saturation 模块设置

Parameters
Numerator:
[1 2]
Denominator:
[1 5 5]
Absolute tolerance:
auto

图 10.5 模块 Transfer Fcn 模块设置

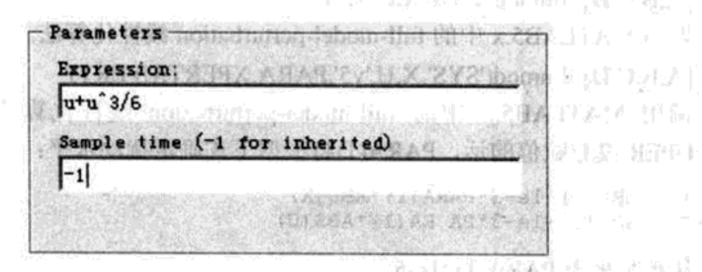


图 10.6 模块 Fcn 模块设置

使用下面的命令来获取系统默认的状态的工作点：

```
>> [x,u,y,dx]=trim(' model10to02')
x =
     0
     0
     0
u =
     0
y =
     0
dx =
     0
     0
```

```
    0
```

还可以获得阶跃输入下的工作点，可以通过下面的命令：

```
>> [x,u,y,dx]=trim(' model10to02',[],1)
x =
   -0.1664
   -0.0000
    0.0666
u =
    1
y =
   -0.1664
dx =
  1.0e-016 *
         0
   -0.5551
   -0.0000
```

2. 连续系统和离散系统的线性化模型

- [A,B,C,D]=linmod('SYS')
  得到默认设置下非线性常微分方程的线性状态方程；
- [A,B,C,D]=linmod('SYS',X,U)
  得到指定状态向量 X 和输入 U 条件下的线性模型方程；
- [A,B,C,D]=linmod('SYS',X,U,PARA)
  允许设置一个参数向量， PARA(1) 设置扰动程度，系统是时间参数 PARA(2)的函数，设定线性化的时间，PARA(2)为 t 的函数，默认 t=0，设置 PARA(3)=1，用来去除没有在输入输出中出现的状态变量。
- [A,B,C,D]=linmod('SYS',X,U,'v5')
  调用 MATLAB5.x 中的 full-model-perturbation 线性化算法。
- [A,B,C,D]=linmod('SYS',X,U,'v5',PARA,XPERT,UPERT)
  调用 MATLAB5.x 中的 full-model-perturbation 线性化算法。如果 XPERT 和 UPER 没有赋值的话，PARA(1)将根据下式确定扰动水平：

```
XPERT= PARA(1)+1e-3*PARA(1)*ABS(X)
UPERT= PARA(1)+1e-3*PA RA(1)*ABS(U)
```

默认扰动水平为 PARA(1)=1e-5

如果向量 XPERT 和 UPERT 都赋值，就会作为扰动水平。

还有一个功能类似的函数，linmod2 函数。

离散系统用 dlinmod 函数，方法和上面一样。

3. 模型线性化实例

【例 1003】求非线性系统$\begin{cases}\dot{x}_1=-x_1^2+x_1-x_2^2+10\\ \dot{x}_2=-2x_1^2-x_2\end{cases}$在原点坐标处的线性化模型。

(1) 利用前面的方法手工计算矩阵 A、B、C、D；

$$A=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix},\quad B=\begin{bmatrix}0\\ 0\end{bmatrix},\quad C=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix},\quad D=\begin{bmatrix}0\\ 0\end{bmatrix}$$

(2) 创建 Simulink 模型，保存为 model10to03.mdl，如图 10.7 所示。

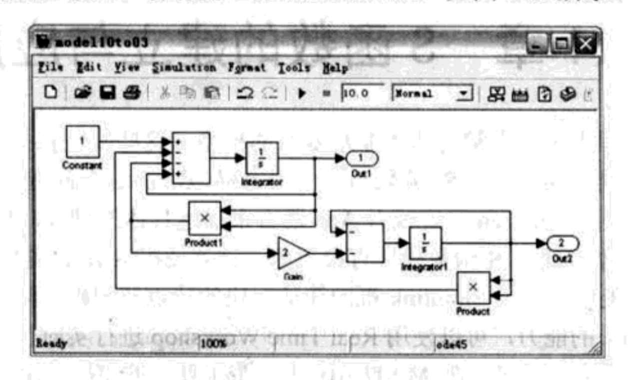


图 10.7　Simulink 模型

(3) 用命令求坐标原点处的线性化模型，可以发现与手工计算一致。

```
>> [A,B,C,D]=linmod(' model10to03')
A =
    1.0000         0
         0   -1.0000
B =
   Empty matrix: 2-by-0
C =
    1.0000         0
         0    1.0000
D =
   Empty matrix: 2-by-0
```

(4) 用命令求点[1,0.5]坐标处的线性化模型。

```
>> [A1,B1,C1,D1]=linmod(' model10to03',[1,1])
A1 =
   -1.0000   -2.0000
   -4.0000   -1.0000
B1 =
   Empty matrix: 2-by-0
C1 =
    1.0000         0
         0    1.0000
D1 =
   Empty matrix: 2-by-0
```

(5) 判断系统的稳定性。

```
>> EngA=eig(A),EngA1=eig(A1)
EngA =
    1.0000
   -1.0000
EngA1 =
   -3.8284
    1.8284
```

可以从特征值中可以看出，系统在[0,0]和[1,1]处都是不稳定的。

第 11 章　S 函数的建立与应用

在实际建模过程当中，常常会遇到非常复杂的模型，这些模型直接用 Simulink 创建显得非常复杂或者是不可能的，S 函数却可以很容易地解决这个问题。用 MATLAB 语言，C 语言，C++语言，Fortran 语言或者 Ada 语言来描述具体的过程，构成 S 函数模块，然后在 Simulink 模型中通过 S 函数模块直接调用。这就使得 S 函数成为 Simulink 展现魅力的一个亮点，完美地结合了 Simulink 框图简洁明快的特点和编程灵活的优点。S 函数增强和扩展了 Simulink 的能力，可以使用 Real Time Workshop 进行实时仿真。

S 函数有固定的编写格式，在 MATLAB 中自带了默认的模板，用户只需要按其要求填写或改写相关部分。直接利用 MATLAB 语言编写的 S 函数不需要编译就可以直接调用，而其他的 C，C++，Fortran 和 Ada 都需要通过编译之后才能够被调用。其实 Simulink 的许多模块所包含的算法就是用 S 函数编写的，用户也可以仿造这个功能自己编写特定功能的 S 函数封装到模块当中。

本章主要内容包括：

- 何为 S 函数
- 在模型中使用 S-Functions
- S 函数工作原理
- M 文件 S 函数的编写
- M 文件 S 函数模板
- M 文件 S 函数简单实例
- 连续，离散和混合系统 M 文件 S 函数
- C 语言编写 S 函数模板

11.1　S 函数介绍

S 函数就是 S-Functions，是 system-Functions 的缩写。当 MATLAB 所提供的模型不能完全满足用户要求时，就可以通过 S 函数提供给用户自己编写程序来满足自己要求模型的接口。S 函数可以用 MATLAB, C, C++, Ada 和 Fortran.编写。C, C++, Ada, and Fortran S-Functions 需要编译为 MEX 文件，就和其他的 MEX 文件一样，Simulink 可以随时动态的调用这些文件。

S 函数使用的是一种比较特殊的调用格式，可以和 Simulink 求解器进行交互式操作，这种交互式就与 Simulink 求解器和内置固有模块交互式操作相同。S-Functions 功能非常全面，适用于连续、离散以及混合系统。

S 函数允许用户向模型中添加自己编写的模块，只要按照一些简单的规制，就可以在 S-Functions 添加设计算法。在编写好 S-Functions 之后就可以在 S-Functions 模块中添加相应的函数名，也可以通过封装技术来定制自己的交互界面。

可以在 S 函数中使用 Real-Time Workshop。可以编写一个目标语言编译器(Target Language Compiler)，用 Real Time Workshop 产生 S-Functions 代码。

可以从以下几个角度简单地理解 S 函数：

- S 函数为 Simulink 的系统函数。
- 能够响应 Simulink 求解其命令的函数。
- 采用非图形化的方法实现 Simulink 模块功能的函数。
- 通过编写 S 函数，可以自己开发相应的 Simulink 函数。
- 可以同已经存在的函数一起进行仿真。
- 将一个复杂的系统，通过文本方式输入到系统当中。
- 扩展 Simulink 功能：M 文件 S 函数可以扩展图形功能。C MEX S 函数可以提供与操作系统的接口。
- S 函数的语法结构是为实现一个动态系统而设计的，有相应的模板可以使用。

11.2　在模型中使用 S-Functions

11.2.1　S 函数的调用

为了在 Simulink 中使用 S-Functions，必须从 Simulink 中的 User-Defined Functions 模块库中向 Simulink 模型文件窗口中拖放 S-Functions 模块。然后在 S-Functions 模块的对话框中的 S-Functions name 框中输入 S 函数的文件名。

【例 1101】调用 S 函数

这里只简单地说明如何调用，不求运行的结果。具体如图 11.1 所示。

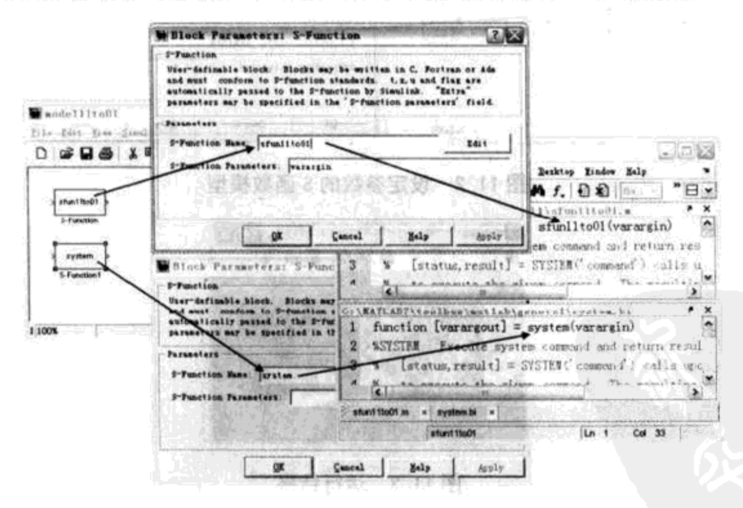


图 11.1　S-Functions 模块、S-Functions 模块参数对话框和 S-Functions 源文件之间的关系图

操作步骤如下：

(1) 新建 Simulink 模块，保存文件名为 Sfun11to01.mdl。

(2) 从 Simulink 中 User-Defined Functions 模块库拖放两个 S-Fucntion 模块到模型

窗口。

(3) 打开两个模块的参数对话框，在 S-Functions name 框中填写相应的 S 函数及输入参数，然后单击 OK 按钮，设置完成。

在这个图形中有两个 S-Functions 模块，每个模块应用不同的文件。两个模块当然也可以引用同一个 S 函数，这个 S 函数可以是 C MEX 文件，也可以是 M 文件。如果 C MEX 文件和 M 文件同名，则 C MEX 文件将被优先考虑。

S 函数模块的 S-Functions parameters 参数框可以设定特定的参数，可以通过这些参数将数据传入到 S 函数当中。但是在使用这个参数框之前，必须知道 S 函数所需要的参数数量以及顺序。如果不知道 S 函数参数形式，可以询问 S 函数的作者、帮助文档以及查看原代码。S-Functions parameters 参数框中的参数输入可以是参数，也可以是模型工作空间中的变量名，以及是 MATLAB 表达式。如果有多个参数，必须按顺序用英文逗号隔开。

【例 1102】使用一个 S 函数，并设定参数。

操作步骤如下：

(1) 建立模型如图 11.2 所示。

(2) 双击 S-Functions 模块，弹出参数对话框，在 S-Functions Name 文本框中输入 limintm，在 S-Functions parameters 文本框中输入“2,3,2.5”，然后单击 OK 按钮。

(3) 运行 Simulink 模型。

(4) 结果如图 11.3 所示。

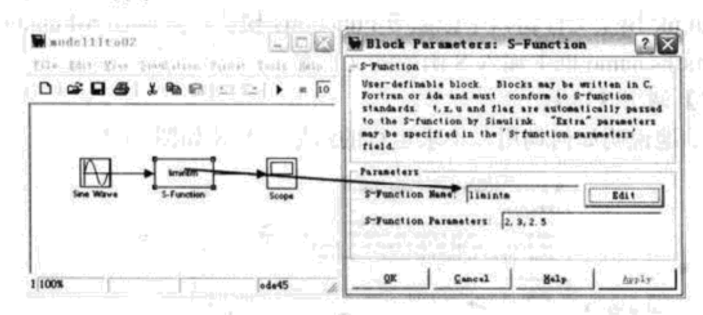


图 11.2　设定参数的 S 函数模型

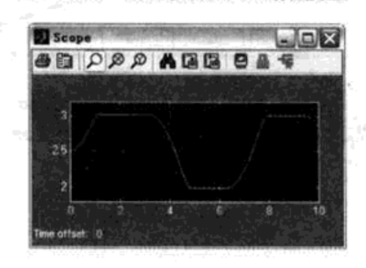


图 11.3　运行结果

这个实例中使用了 limintm 函数，是一个调用采样的 S 函数 Simulink 模型，函数的原代码在 toolbox/simulink/blocks 文件夹中。limintm 函数接受 3 个参数：下界、上界和初始值。如果积分在上界与下界之间，则它的输出是输入信号的积分，如果积分值大于上界，或者小于下界，那么就分别保持上、下界值。此例中下界、上界、初始条件分别是 2、

3、2.5。

limintm 程序原代码系统自带了，用户可以直接在参数对话框中添加，具体程序原代码稍作说明如下：

```
function [sys,x0,str,ts]=limintm(t,x,u,flag,lb,ub,xi)
%   LIMINTM 执行受限积分
%   是一个连续 M 文件的 S 函数执行一个连续受限积分的例子。
%   其中输出受到上界(UB)和下界(LB)的限制
%   系统的初始值为(XI).
%   关于 S 函数模板可参照 sfuntmpl.m
switch flag
  % 初始化 %
  case 0
    [sys,x0,str,ts] = mdlInitializeSizes(lb,ub,xi);
  % 求导 %
  case 1
    sys = mdlDerivatives(t,x,u,lb,ub);
   % 更新 %
  case {2,9}
    sys = []; % do nothing
  % 输出%
  case 3
    sys = mdlOutputs(t,x,u);
  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end
% limintm 结束
%==============================================================
% mdlInitializeSizes% 向 S 函数返回大小，初始条件和样本时间
%==============================================================
function [sys,x0,str,ts] = mdlInitializeSizes(lb,ub,xi)
sizes = simsizes;
sizes.NumContStates  = 1;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
str = [];
x0  = xi;
ts  = [0 0];   % sample time: [period, offset]
% mdlInitializeSizes 结束
%==============================================================
% mdlDerivatives% 对连续系统进行求导
function sys = mdlDerivatives(t,x,u,lb,ub)
if (x <= lb & u < 0)  | (x>= ub & u>0 )
  sys = 0;
else
  sys = u;
end
% mdlDerivatives 结束
%==============================================================
% mdlOutputs% S 函数输出
function sys = mdlOutputs(t,x,u)
sys = x;
% mdlOutputs 结束
```

11.2.2 S 函数所起的作用

最常使用的就是用(最常用的是用)S 函数来创建一个用户自定义的 Simulink 模块。除此之外，还可以用 S 函数来实现以下几个方面的功能：

- 向 Simulink 模型中增加一个通用目的的模块。
- 使用 S 函数的模块来充当硬件的驱动。
- 在仿真中嵌入已经存在的 C 代码。
- 将系统表示成一系列的数学方程。
- 在 Simulink 中使用动画。

使用 S 函数的一个优点就是，用户可以建造一个通用目的的模块，在一个模型中可以多次使用，每一个模块可以有不同的参数。

11.3 S 函数工作原理

了解 S 函数的工作原理对编写 S 函数，以及整个 Simulink 仿真过程都是非常有帮助的。在创建 S 函数之前，有必要知道 S 函数是如何工作的，接下来就要知道 Simulink 是如何进行模型仿真的，最后还要知道模块的数学模型是什么？

11.3.1 模型的数学模型

本节主要讲述模块的输入、状态和输出的关系。

Simulink 模块一般由若干输入，若干状态和若干输出组成，其中输出是时间、状态和输入的函数，关系如图 11.4 所示。

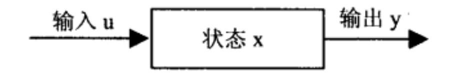


图 11.4 输入、状态和输出关系图

可以通过下面几个数学表达式来描述上述过程，具体如下：

输出： $y = f(x,u,t)$

状态： $\dot{x} = g(x,u,t)$

11.3.2 仿真过程

Simulink 模型的处理过程主要有两个过程，具体如下：

1. 初始化阶段

这时候模块的所有参数都将确定，主要完成以下几个过程：

(1) 传递参数给 MATLAB 进行求值。

(2) 得到的数值作为实际的参数使用。

(3) 展开模型的层次，每个子系统被它们所包含的模块替代。

(4) 检查信号的连接。

(5) 确定状态初值和采样时间。

2. 运行阶段

仿真开始运行，仿真过程是由求解器和 Simulink 引擎交互控制的。求解器的作用是传递模块的输出，对状态导数进行积分，并确定采样时间，周而复始，一直到仿真结束。

仿真运行阶段的工作可以概括为：

(1) 计算输出。

(2) 更新离散状态。

(3) 计算连续状态，连续状态。

(4) 计算输出，过零可能被激活。

图 7.5 描述了整个仿真过程。

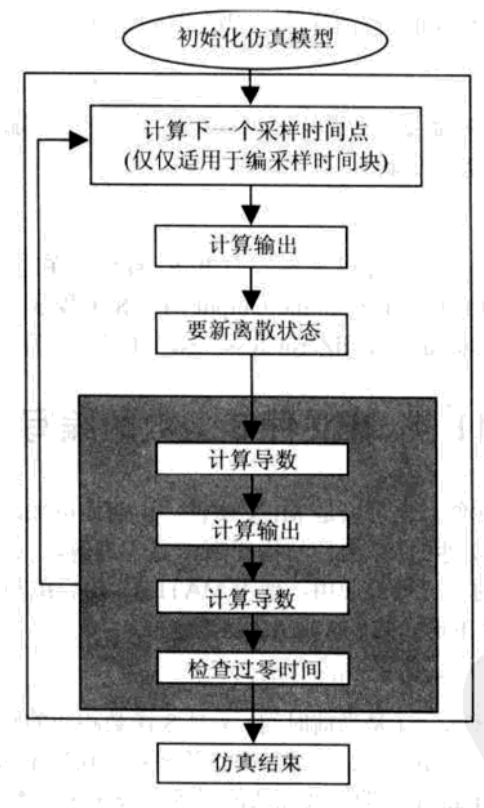


图 11.5 仿真过程

11.3.3 S 函数回调方法

一个 S 函数有一系列 S 函数回调方法，这些回调方法在仿真的每一步是必需的。在模型的仿真期间，在仿真的每一步，Simulink 为每一个 S 函数模块调用适当的方法。S 函

数方法主要完成以下几个方面工作：

1. 初始化

在仿真之前，Simulink 在这个阶段初始化 S 函数，主要有以下几个过程：

(1) 初始化结构体 SimStruct，它包含了 S 函数的所有信息。

(2) 设置输入输出端口数。

(3) 设置采样时间。

(4) 分配存储空间。

2. 计算下一个采样时间点

在创建一个变时间样本时间模块时，这个步骤计算下一个样本的时间点，即计算下一步的仿真步长。

3. 计算下一个时间步的输出

当这一步完成后，此模块的所有端口对于当前时间步都是合法的。

4. 更新状态

该过程在每一个步长处都要执行一次，可以在这个过程中添加每一个仿真都需要更新的内容，此过程是离散状态的更新。

5. 积分

用于连续状态的求解和非采样过零点，如果 S 函数存在连续状态，Simulink 就在 minor step time 内调用 mdlDdrivatives 和 mdlOutput 两个 S 函数过程。如果存在非采样过零点，Simulink 将调用 mdlOutput 和 mdlZeroCrossings，过程一定为过零点。

11.4 M 文件 S 函数的编写

S 函数既可以是 M 文件，也可以是 MEX 文件。由前面可知，S 函数无非是由一些仿真功能模块组成的。M 文件的 S 函数结构明晰，易于理解，书写方便，而且可以调用 MATLAB 丰富的函数，对于一般的应用，使用 MATLAB 语言编写 S 函数就够用了。

M 文件的 S 函数由以下形式的 MATLAB 函数组成：

```
[sys,x0,str,ts]=f(t,x,u,flag,p1,p2,...)
```

其中 f 是 S 函数的文件名，t 是当前时间，x 是 S 函数相应的状态向量，u 是模块的输入，flag 是所要执行的任务，p1, p2, …都是模块的参数。在模型仿真的过程中，Simulink 不断地调用函数 f，通过 flag 来说明所要完成的任务。每次 S 函数执行任务，都将以特定结构返回结果。

M 文件的 S 函数和前面介绍的 S 函数流程一样，调用顺序通过标志 Flag 来控制。如图 11.6 所示，给出了一个仿真阶段的标志值、变量值及其对应仿真过程，具体说明见表 11.1 所示。

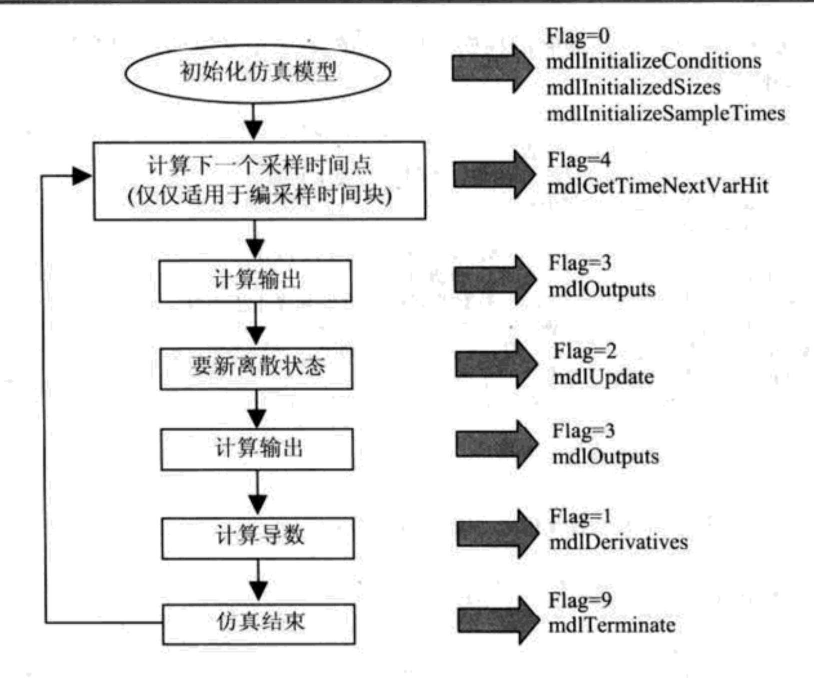


图 11.6 M 文件 S 函数流程

表 11.1 仿真过程

仿真阶段	S 函数过程	Flag
初始化	mdlInitializeSizes	flag=0
计算下一个采样时间	mdlGetTimeOfNextVarHit	flag=4
计算输出	mdlOutputs	flag=3
更新离散状态	mdlUpdate	flag=2
计算导数	mdlDerivatives	flag=1
计算任务结束	mdlTerminate	flag=9

11.5 M 文件 S 函数模板

Simulink 为编写 S 函数提供了各种模板文件，其中定义了 S 函数完整的框架结构，用户可以根据自己的需要加以修剪。推荐在编写 M 文件 S 函数时，使用 S 函数模板文件 sfunmpl.m，在文件夹 matlabroot/toolbox/simulink/中，这个文件中包含了一个完整的 M 文件 S 函数。其中包括一个主函数和若干子函数，每一个子函数都对应一个 flag。主函数来调用子函数，子函数就成为 S 函数回调函数。在主函数内根据 flag 变量，有一个开关转移结构(Switch-Case)根据标志将执行相应的流程转到对应的子函数。

用户可以在 MATLAB 命令窗口输入如下命令：

```
>>edit sfuntmpl
```

用户也可以在文件夹中直接双击 sfuntmpl.m 文件，文件夹为 matlabroot/toolbox/simulink/。

为了描述方便，在此给出简略的模板文件，并配置了中文说明。如果要了解更多请打开原版模板文件。

```
function [sys,x0,str,ts] = sfuntmpl(t,x,u,flag)
% SFUNTMPL 是 M-文件 S 函数模板
% 通过剪裁，用户可以生成自己的 S 函数，不过一定要重新命名
% 利用 S 函数可以生成连续、离散混合系统等，实现任何模块的功能
% M-文件 S 函数的语法为：
% [SYS,X0,STR,TS] = SFUNC(T,X,U,FLAG,P1,...,Pn)
% 用户切勿改动输出参数的顺序、名称、和数目
% 输入参数的数目不能小于 1，这四个参数的名称和排列顺序不能改动
% 用户可以根据自己的要求添加额外的参数，位置依次为第 5，6，7，8，9 等。
% S 函数的 flag 参数是一个标记变量，具有 6 个不同值，分别为 0，1，2，3，4，9
% flag 的 6 个值分别指向 6 个不同的子函数
% flag 所指向的子函数也成为回调方法(Callback Methods)
switch flag,
% 初始化，调用“模块初始化”子函数 %
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
% 连续状态变量计算，调用“计算模块导数”子函数 %
  case 1,
    sys=mdlDerivatives(t,x,u);
% 更新，调用“更新模块离散状态”子函数 %
  case 2,
    sys=mdlUpdate(t,x,u);
% 输出，调用“计算模块输出”子函数%
  case 3,
    sys=mdlOutputs(t,x,u);
%计算下一时刻采样点调用“计算下一个采样时点”子函数%
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);
%结束，调用“结束仿真”子函数%
  case 9,
    sys=mdlTerminate(t,x,u);
%其他的 flag%
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
% end sfuntmpl
%=============================================================
%  “模块初始化”子函数
%  返回大小、初始条件和样本
function [sys,x0,str,ts]=mdlInitializeSizes
%  调用 simsizes 函数，返回规范格式的 sizes 构架
%  这条指令不要修改
sizes = simsizes;
%  模块的连续状态个数，0 是默认值
%  用户可以根据自己的要求进行修改
sizes.NumContStates  = 0;
%  模块的离散状态个数，0 是默认值
%  用户可以根据自己的要求进行修改
sizes.NumDiscStates  = 0;
%  模块的输出个数，0 是默认值
%  用户可以根据自己的要求进行修改
sizes.NumOutputs     = 0;
%  模块的输入个数，0 是默认值
%  用户可以根据自己的要求进行修改
sizes.NumInputs      = 0;
%  模块中包含的直通前向馈路个数，1 是默认值
%  用户可以根据自己的要求进行修改
```

```
sizes.DirFeedthrough = 1;
%  模块中采样时间的个数，1 是默认值，至少需要一个样本时间
%  用户可以根据自己的要求进行修改
sizes.NumSampleTimes = 1;
%  初始化后的构架 sizes 经 simsizes 函数处理后向 sys 赋值
%  这条指令不要修改
sys = simsizes(sizes);
%  给模块初始值变量赋值，[ ]是默认值
%  用户可以根据自己的要求进行修改
x0  = [];
%  系统保留变量
%  切勿改动，保持为空
str = [];
% “二元对”描述采样时间及偏移量。[0 0]是默认值
%    [0 0]适用于连续系统
%    [-1 0]则表示该模块采样时间继承其前的模块采样时间设置
ts  = [0 0];
%==============================================================
function sys=mdlDerivatives(t,x,u)

%  此处填写计算导数向量的指令
%  []是模块的默认
%  用户必须把算得的离散状态向量赋给 sys
sys = [];
%==============================================================
function sys=mdlUpdate(t,x,u)
%  此处填写计算离散状态向量的指令
%  []是模块的默认
%  用户必须把算得的离散状态向量赋给 sys
sys = [];
%==============================================================
function sys=mdlOutputs(t,x,u)
%  此处填写计算模块输出向量的指令
%  []是模块的默认
%  用户必须把算得的离散状态向量赋给 sys
sys = [];
%==============================================================
function sys=mdlGetTimeOfNextVarHit(t,x,u)
%  该子函数仅在“采样时间”情况下使用
%  sampleTime = 1 是模板默认设置，表示在当前时间 1 秒后再调用本模块
%  用户可以更具自己的要求修改
sampleTime = 1;
%  将计算得到的下一采样时刻赋给 sys
%  切勿改动
sys = t + sampleTime;
%==============================================================
function sys=mdlTerminate(t,x,u)
%  模板默认设置，一般情况不要改动
sys = [];
```

说明：

- 模板文件输入的参数含义如下：
  - t ——当前时刻，是采用绝对计量的时间值，是从仿真开始模型运行时间的计量值。
  - x ——模块的状态向量，包括连续状态向量和离散状态向量。
  - u ——模块的输入向量。
  - flag ——执行不同操作的标记变量。

- 在 Simulink 仿真过程中，各步骤的不同功能之间的转换，是通过 flag 标记来实现的，flag 取不同值时所代表的功能如表 11.2 所示。
- 模板文件中的 case 不一定都是必要的，有的情况下，可以进行裁剪，例如当模块不采用变采样时，case4 和相应的子函数 mdlGetTimeOfNextVarHit 就可以删除。
- 用户对 S 函数模板进行裁剪，不能修改输入和输出参数的名称、顺序，不过可以增加输入参数的数量。

表 11.2　Flag 值的功能

Flag 值	功　能
0	模块初始化
1	计算模块导数
2	更新模块离散状态
3	计算模块输出
4	计算下一个采样时点
9	结束仿真

11.6　M 文件 S 函数简单实例

M 文件 S 函数的编写，除了所必需的参数之外，还可以添加自定义的参数，这些参数需要在 S 函数的输入参数中列出。首先对主函数要作适当的修改，这样用户就可以将参数传递到子函数中。同样子函数的定义同时也要进行修改，使得子函数能够接受用户参数。在编写 S 函数时，要注意这些参数所起的作用。在调用 S-Functions 模块时，记住在参数对话框中输入这些自定义的参数。

【例 1103】使用 S 函数实现增益，使得 y=2*u。

步骤如下：

(1) 对 M 文件 S 函数的主函数进行修改，增加新的函数。

```
function [sys,x0,str,ts] = mtimestwo(t,x,u,flag)
```

(2) 修改初始化。

```
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = -1;  % dynamically sized
sizes.NumInputs      = -1;  % dynamically sized
sizes.DirFeedthrough = 1;   % has direct feedthrough
sizes.NumSampleTimes = 1;
```

(3) mdlOutputs 子函数的定义也做相应的修改，将增益作为参数输入。

```
function sys=mdlOutputs(t,x,u)
sys =2*u;
```

(4) 建立模型如图 11.7 所示，保存文件名为 model11to03.mdl。

(5) S-Functions 模块的参数设置如图 11.8 所示。
(6) Sine Wave 模块中将 Frequency 设置为 2。
(7) 运行仿真，结果如图 11.9 所示。

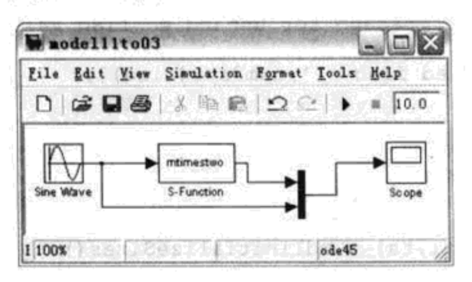


图 11.7 模型 model11to03 仿真图

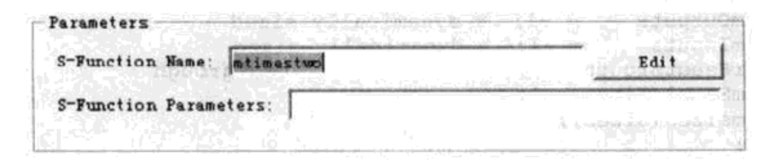


图 11.8 S-Functions 模块的参数设置

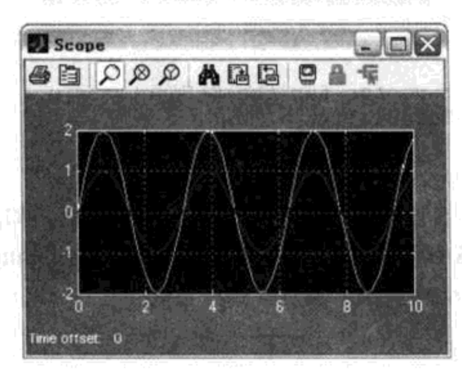


图 11.9 仿真结果

完整程序代码如下：

```
function [sys,x0,str,ts] = sfun11to02(t,x,u,flag)
%TIMESTWO S-Functions whose output is two times its input.
%   This M-file illustrates how to construct an M-file S-Functions that
%   computes an output value based upon its input.  The output of this
%   S-Functions is two times the input value:
%     y = 2 * u;
switch flag,
  % Initialization %
  % Initialize the states, sample times, and state ordering strings.
  case 0
    [sys,x0,str,ts]=mdlInitializeSizes;

  % Outputs %
  % Return the outputs of the S-Functions block.
  case 3
    sys=mdlOutputs(t,x,u);
```

```
  % Unhandled flags %
  case { 1, 2, 4, 9 }
    sys=[];
  % Unexpected flags (error handling)%
  % Return an error message for unhandled flag values.
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
% end timestwo
%==============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-
Functions.
function [sys,x0,str,ts] = mdlInitializeSizes()
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = -1;  % dynamically sized
sizes.NumInputs      = -1;  % dynamically sized
sizes.DirFeedthrough = 1;   % has direct feedthrough
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
str = [];
x0  = [];
ts  = [-1 0];   % inherited sample time
% end mdlInitializeSizes
%==============================================================
% mdlOutputs
% Return the output vector for the S-Functions
function sys = mdlOutputs(t,x,u)
sys = u * 2;
% end mdlOutputs
```

【例 1104】本例使用 S 函数来对一个单摆系统进行仿真，如图 11.10 所示。只要演示以下 3 个方面：①利用 S 函数对单摆系统进行建模；②利用 Simulink 进行仿真，研究单摆的位移；③利用 S 函数动画模块来演示单摆的运动。

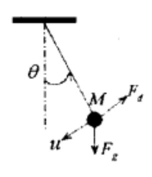


图 11.10 单摆示意图

步骤如下：

(1) 单摆的动力学方程为：$M\ddot{\theta}+K_d\dot{\theta}=u-F_g\sin(\theta)$

其中：u 为实施在单摆上的外力，K_d 为阻尼系数；F_g 为重力。

(2) 将系统动力学方转化为状态方程：

令 $x_1=\theta$，$x_2=\dot{\theta}$，

动力学方程变为：

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix}=\begin{bmatrix}x_2\\ -K_d x_2+u-F_g\sin(x_1)\end{bmatrix}$$

(3) 根据状态方程，在模板的基础上编写 S 函数，保存为 Sfunction1104.m。

```
function [sys,x0,str,ts] = sfunction0704(t,x,u,flag,damp,grav,ang,m)
switch flag,
  case 0,       % Initialization
    [sys,x0,str,ts]=mdlInitializeSizes(ang);
  case 1,       % Derivatives
    sys=mdlDerivatives(t,x,u,damp,grav,m);
  case 2,       % Update
    sys=mdlUpdate(t,x,u);
  case 3,       % Outputs
    sys=mdlOutputs(t,x,u);
  case 9,       % Terminate
    sys=mdlTerminate(t,x,u);
  otherwise     % Unexpected flags
    error(['Unhandled flag = ',num2str(flag)]);
end
% ------ mdlInitializeSizes -----------------------
function [sys,x0,str,ts]=mdlInitializeSizes(ang)
sizes = simsizes;                % 固定格式
sizes.NumContStates  = 2;        % 连续状态变量数量为 2
sizes.NumDiscStates  = 0;        % 离散状态变量数量为 0
sizes.NumOutputs     = 1;        % 有 1 个输出
sizes.NumInputs      = 1;        % 有 1 个输入
sizes.DirFeedthrough = 0;        % 输入输出间不存在直接比例关系
sizes.NumSampleTimes = 1;        % 只有 1 个采样时间
sys = simsizes(sizes);           % 固定格式
x0  = ang;                       % 初始状态值
str = [];                        % 固定格式
ts  = [0,0];                     % 该取值对应纯连续系统
% -----mdlDerivatives ------------------------------
function sys=mdlDerivatives(t,x,u,damp,grav,m)
% 系统状态方程
dx(1)=x(2);
dx(2)=-damp*x(2)-m*grav*sin(x(1))+u;
sys =dx;                    % 把计算得到的导数向量向 sys 赋值
% -----mdlUpdate ----------------------------------
function sys=mdlUpdate(t,x,u)
sys = [];                   % 不做任何修改
% -----mdlOutputs ---------------------------------
function sys=mdlOutputs(t,x,u)
sys = x(1);                 % 把计算得到的变来能够赋给 sys
% -----mdlTerminate -------------------------------
function sys=mdlTerminate(t,x,u)
sys = [];                   % 保持默认设置
```

(4) 搭建 Simulink 模型，保存为 model11to04.mdl，如图 11.11 所示。

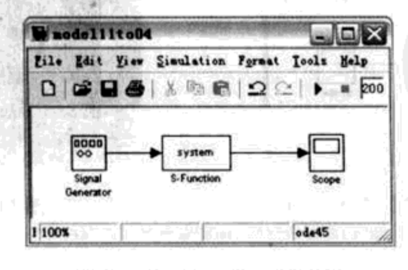


图 11.11　Simulink 模型图

(5) 对 Simulink 模型进行参数设置。

- 双击 S-Functions 模块，弹出参数对话框，在 S-Functions Name 文本框中输入 sfun11to03，在 S-Functions Parameters 文本框中输入“damp,grav,ang,m”，如图 11.12 所示。

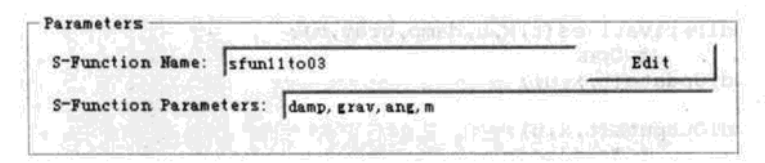


图 11.12 S-Functions 模块参数对话框设置

- 双击 Signal Generator 模块，弹出参数对话框，在 Wave form 下拉列表框中选择 square 选项，Amplitude 为 1，Frequency 为 0.02，Units 为 Hertz，如图 11.13 所示。

Parameters
Wave form: square
Time (t): Use simulation time
Amplitude:
1
Frequency:
0.02
Units: Hertz
Interpret vector parameters as 1-D

图 11.13 Signal Generator 模块参数对话框设置

(6) 设置 Simulink 模型的仿真运行时间为 200 秒。

(7) 设置完成后，Simulink 模型中的 S-Functions 模块就会作相应的变化，结果如图 11.14 所示。

(8) 在 MATLAB 窗口中输入：

```
>> damp=0.8;grav=9.8;ang=[0;0];m=0.2;
```

(9) 运行仿真，结果如图 11.15 所示。

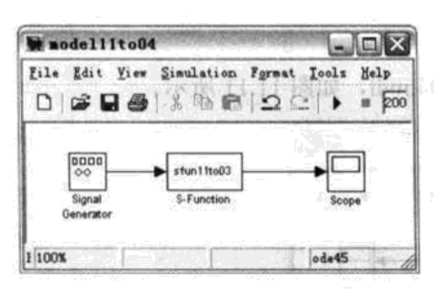


图 11.14 编辑完成后的仿真模型图

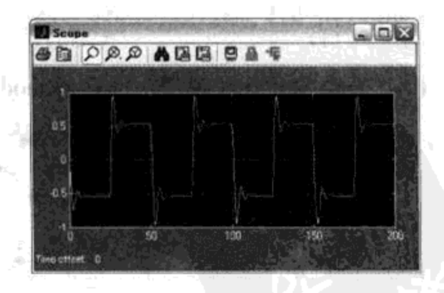


图 11.15 仿真结果

(10) 引进单摆动画模块。

- 将 model11to04.mdl 另存为 model11to05.mdl。
- 打开 toolbox\simulink\simdemos\simgeneral 子目录下的 simppend.mdl 模型。

- 将其中的 Animation Function 模块，Pivot point for pendulum 以及 x&theta 模块复制到 model11to05.mdl 模型窗口，进行相应的连接，模型如图 11.16 所示。

(11) 再次进行仿真，结果如图 11.17 所示。

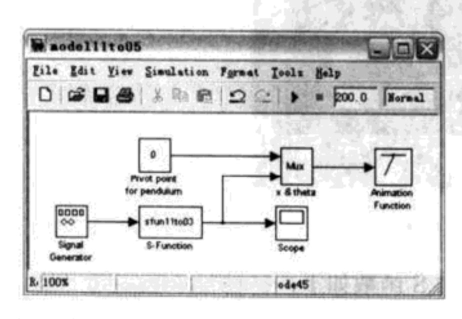




图 11.16　连接动画显示模块的 Simulink 模型

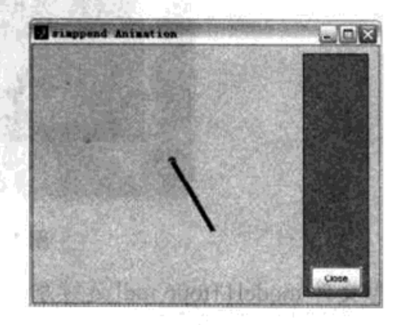


图 11.17　动画输出结果

11.7　连续，离散和混合系统 M 文件 S 函数

11.7.1　连续系统

【例 1105】编写一个 S 函数代替连续系统状态方程模块的功能，实现

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases}$$

其中 $A=\begin{bmatrix} -0.09 & -0.01 \\ 1 & 0 \end{bmatrix}$，$B=\begin{bmatrix} 1 & -7 \\ 0 & -2 \end{bmatrix}$，$C=\begin{bmatrix} 0 & 2 \\ 1 & -5 \end{bmatrix}$，$D=\begin{bmatrix} -3 & 0 \\ 1 & 0 \end{bmatrix}$。

步骤如下：

(1) 建立模型，Simulink 模型如图 11.18 所示，保存为 model11to06.mdl。

- 模块 Sine Wave 和模块 Random Number 都采用默认设置。
- S-Functions 在 S-Functions name 栏设置为 csfunc。

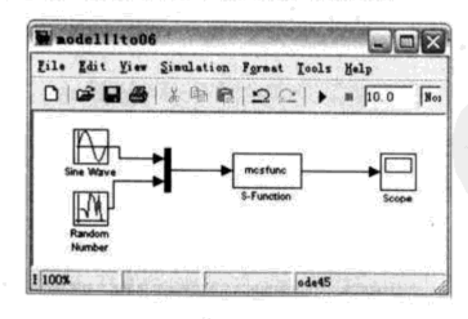


图 11.18　Simulink 模型

(2) 运行模型。

(3) 运行结果如图 11.19 所示。

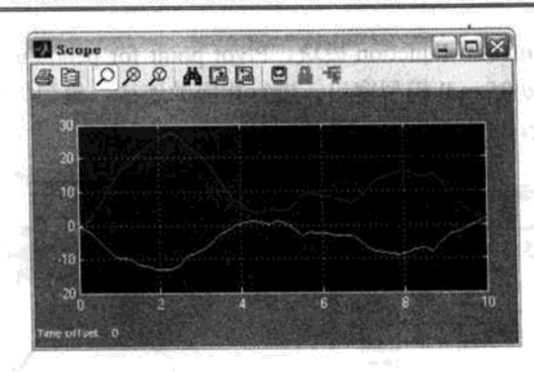


图 11.19 仿真结果

模型文件 model11to06.mdl 连续系统的 M 文件 S 函数如下：

```
function [sys,x0,str,ts] = sfun11to04(t,x,u,flag)
%CSFUNC An example M-file S-Functions for defining a continuous system.
%   Example M-file S-Functions implementing continuous equations:
%      x' = Ax + Bu
%      y  = Cx + Du
A=[-0.09   -0.01
   1          0];
B=[ 1   -7
   0   -2];
C=[ 0    2
   1   -5];
D=[-3    0
   1    0];
switch flag,
  % Initialization %
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D);

 % Derivatives %
  case 1,
    sys=mdlDerivatives(t,x,u,A,B,C,D);

  % Outputs %
  case 3,
    sys=mdlOutputs(t,x,u,A,B,C,D);
  % Unhandled flags %
  case { 2, 4, 9 },
    sys = [];

  % Unexpected flags %
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
% end csfunc

%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-
Functions.
function [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D)
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
```

```
sizes.NumOutputs     = 2;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = zeros(2,1);
str = [];
ts  = [0 0];

% end mdlInitializeSizes
%=============================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
function sys=mdlDerivatives(t,x,u,A,B,C,D)
sys = A*x + B*u;
% end mdlDerivatives
%=============================================================
% mdlOutputs
function sys=mdlOutputs(t,x,u,A,B,C,D)
sys = C*x + D*u;
% end mdlOutputs
```

11.7.2 离散系统

【例 1106】编写一个 S 函数代替连续系统状态方程模块的功能，实现

$$\begin{cases} x(n+1) = Ax(n) + Bu(n) \\ y(n) = Cx(n) + Du(n) \end{cases}$$

其中 $A=\begin{bmatrix} -1.3839 & -0.5097 \\ 1 & 0 \end{bmatrix}$，$B=\begin{bmatrix} -2.5559 & 0 \\ 0 & 4.2382 \end{bmatrix}$，$C=\begin{bmatrix} 0 & 2.0761 \\ 0 & 7.7891 \end{bmatrix}$，$D=\begin{bmatrix} -0.8141 & -2.9334 \\ 1.2426 & 0 \end{bmatrix}$。

步骤如下：

(1) 建立模型，Simulink 模型如图 11.20 所示，保存为 model11to07.mdl。

- 模块 Sine Wave 和模块 Random Number 都采用默认设置。
- S-Functions 在 S-Functions name 栏设置为 sfun11to05。

(2) 运行模型。

(3) 运行结果如图 11.21 所示。

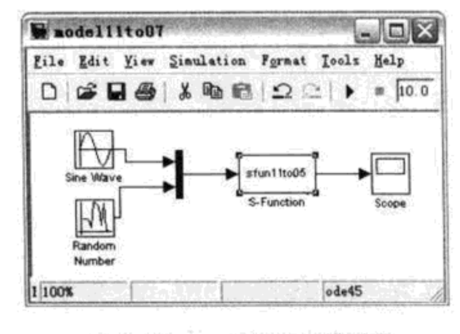


图 11.20 model11to07 模型图

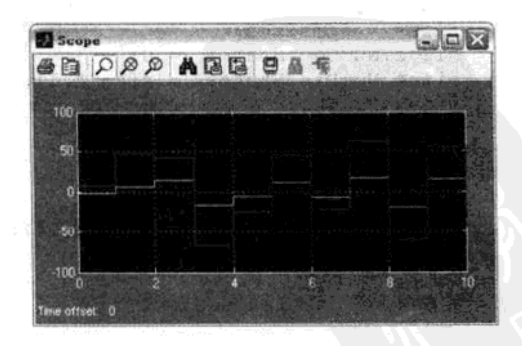


图 11.21 仿真结果

模型中使用的 S 函数如下：

```
function [sys,x0,str,ts] = sfun11to05(t,x,u,flag)
%DSFUNC An example M-file S-Functions for defining a discrete system.
%   Example M-file S-Functions implementing discrete equations:
%      x(n+1) = Ax(n) + Bu(n)
%      y(n)   = Cx(n) + Du(n)

% Generate a discrete linear system:
A=[-1.3839   -0.5097
    1.0000         0];
B=[-2.5559         0
        0    4.2382];
C=[      0    2.0761
        0    7.7891];
D=[   -0.8141   -2.9334
      1.2426         0];
switch flag,

  % Initialization %
  case 0,
    [sys,x0,str,ts] = mdlInitializeSizes(A,B,C,D);

  % Update %
  case 2,
    sys = mdlUpdate(t,x,u,A,B,C,D);

  % Output %
  case 3,
    sys = mdlOutputs(t,x,u,A,C,D);

  % Terminate %
  case 9,
    sys = []; % do nothing

  % Unexpected flags %
  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end
%end dsfunc

%=================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-
Functions.
function [sys,x0,str,ts] = mdlInitializeSizes(A,B,C,D)
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = size(A,1);
sizes.NumOutputs     = size(D,1);
sizes.NumInputs      = size(D,2);
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = ones(sizes.NumDiscStates,1);
str = [];
ts  = [1 0];
% end mdlInitializeSizes

%=================================================================
% mdlUpdate
```

```
% Handle discrete state updates, sample time hits, and major time step
% requirements.
function sys = mdlUpdate(t,x,u,A,B,C,D)
sys = A*x+B*u;
%end mdlUpdate

%=============================================================================
% mdlOutputs
% Return the output vector for the S-Functions
function sys = mdlOutputs(t,x,u,A,C,D)
sys = C*x+D*u;
%end mdlOutputs
```

11.7.3　混合系统

混合系统是指系统中既包含连续系统，又包含离散系统。Simulink 根据 flag 的具体值判断系统是计算连续部分还是离散部分，并调用相应的子程序。Simulink 在处理混合系统时将同时调用 S 函数中的 mdlUpdate、mdlOutput 和 mdlGetTimeOfNextWarHit 子程序。

下面通过一个 S 函数来实现如图 11.22 所示模型。

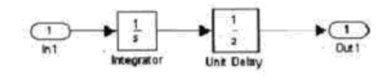


图 11.22　通过 Simulink 所要实现的模型

步骤如下：

(1) 建立模型，Simulink 模型如图 11.23 所示，保存为 model11to08.mdl。

模块 Sine Wave 和模块 Random Number 都采用默认设置。S-Functions 在 S-Functions name 栏设置为 dsfunc。

(2) 运行模型。

(3) 运行结果如图 11.24 所示。

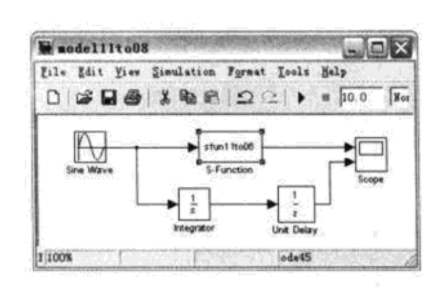


图 11.23　通过模块与 S 函数共同实现的混合模型

图 11.24　混合模型比较结果图

混合模型中使用的 S 函数文件如下：

```
function [sys,x0,str,ts] = sfun11to06(t,x,u,flag)
%MIXEDM An example integrator followed by unit delay M-file S-Functions
%   Example M-file S-Functions implementing a hybrid system consisting
%   of a continuous integrator (1/s) in series with a unit delay (1/z).
%   See sfuntmpl.m for a general S-Functions template.
```

```
% Sampling period and offset for unit delay.
dperiod = 1;
doffset = 0;

switch flag

  % Initialization %
  case 0
    [sys,x0,str,ts]=mdlInitializeSizes(dperiod,doffset);

  % Derivatives %
  case 1
    sys=mdlDerivatives(t,x,u);

  % Update %
  case 2,
    sys=mdlUpdate(t,x,u,dperiod,doffset);

  % Output %
  case 3
    sys=mdlOutputs(t,x,u,doffset,dperiod);

  % Terminate %
  case 9
    sys = [];       % do nothing
  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end
% end mixedm

%==============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-
Functions.
function [sys,x0,str,ts]=mdlInitializeSizes(dperiod,doffset)
sizes = simsizes;
sizes.NumContStates  = 1;
sizes.NumDiscStates  = 1;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 2;
sys = simsizes(sizes);
x0  = ones(2,1);
str = [];
ts  = [0 0;              % sample time
      dperiod doffset];
% end mdlInitializeSizes

%==============================================================
% mdlDerivatives
% Compute derivatives for continuous states.
function sys=mdlDerivatives(t,x,u)
sys = u;
% end mdlDerivatives

%==============================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
function sys=mdlUpdate(t,x,u,dperiod,doffset)
```

```
% next discrete state is output of the integrator
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-8
  sys = x(1);
else
  sys = [];
end
% end mdlUpdate

%=================================================================
% mdlOutputs
% Return the output vector for the S-Functions
function sys=mdlOutputs(t,x,u,doffset,dperiod)
% Return output of the unit delay if we have a
% sample hit within a tolerance of 1e-8. If we
% don't have a sample hit then return [] indicating
% that the output shouldn't change.
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-8
  sys = x(2);
else
  sys = [];
end
% end mdlOutputs
```

11.8 C 语言编写 S 函数模板

11.8.1 C 语言编写 S 函数模板

C MEX 文件的 S 函数与 M 文件的 S 函数在函数结构和子函数方面基本相同。但是 C 语言的灵活性使得 C MEX 能够实现比 M 文件在算法上更强的功能。Simulink 同样为 C MEX 文件的编写提供了模板 sfuntmpl.c，可以用 M 文件编辑器打开。

```
/*
* sfuntmpl_basic.c: Basic 'C' template for a level 2 S-Functions.
*  | See matlabroot/simulink/src/sfuntmpl_doc.c for a more detailed
template |
* -------------------------------------------------------------------------
*/

#define S_FUNCTION_NAME  sfuntmpl_basic
#define S_FUNCTION_LEVEL 2
/*
 * Need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"

/*====================*
 * S-Functions methods *
 *====================*/

/* Function: mdlInitializeSizes ===============================
 * Abstract:
 *   The sizes information is used by Simulink to determine the S-
Functions
 *   block's characteristics (number of inputs, outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S)
```

```
{
    /* See sfuntmpl_doc.c for more details on the macros below */
    ssSetNumSFcnParams(S, 0);  /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        /* Return if number of expected != number of actual parameters */
        return;
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 1);
    ssSetInputPortRequiredContiguous(S, 0, true); /*direct input signal
access*/
    /*
     * Set direct feedthrough flag (1=yes, 0=no).
     * A port has direct feedthrough if the input is used in either
     * the mdlOutputs or mdlGetTimeOfNextVarHit functions.
     * See matlabroot/simulink/src/sfuntmpl_directfeed.txt.
     */
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);
    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    ssSetOptions(S, 0);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_INITIALIZE_CONDITIONS   /* Change to #undef to remove
function */
#if defined(MDL_INITIALIZE_CONDITIONS)
  static void mdlInitializeConditions(SimStruct *S)
  {
  }
#endif /* MDL_INITIALIZE_CONDITIONS */

#define MDL_START  /* Change to #undef to remove function */
#if defined(MDL_START)
  static void mdlStart(SimStruct *S)
  {
  }
#endif /*  MDL_START */

static void mdlOutputs(SimStruct *S, int_T tid)
{
    const real_T *u = (const real_T*) ssGetInputPortSignal(S,0);
    real_T       *y = ssGetOutputPortSignal(S,0);
    y[0] = u[0];
}

#define MDL_UPDATE  /* Change to #undef to remove function */
```

```
#if defined(MDL_UPDATE)
  /* Function: mdlUpdate ======================================
   * Abstract:
   *    This function is called once for every major integration time
step.
   *    Discrete states are typically updated here, but this function is
useful
   *    for performing any tasks that should only take place once per
   *    integration step.
   */
  static void mdlUpdate(SimStruct *S, int_T tid)
  {
  }
#endif /* MDL_UPDATE */

#define MDL_DERIVATIVES  /* Change to #undef to remove function */
#if defined(MDL_DERIVATIVES)

  static void mdlDerivatives(SimStruct *S)
  {
  }
#endif /* MDL_DERIVATIVES */

static void mdlTerminate(SimStruct *S)
{
}

/*=============================================================*
 * See sfuntmpl_doc.c for the optional S-Functions methods *
 *=============================================================*/

/*================================*
 * Required S-Functions trailer *
 *================================*/

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file?
*/
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

11.8.2 C 文件 S 函数倍增实例

下面做一个简单的实例，表示一个增益的功能，别看简单，但包括了 S 函数的基本部分。这是 MATLAB 实例库自带的例子。

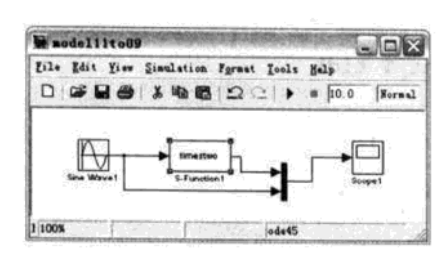


图 11.25 实例仿真模型

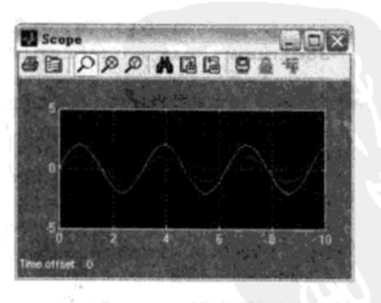


图 11.26 仿真结果

模型中使用的 S 函数 C 文件，文件名为 timestwo.c。

```
/*
 * File : timestwo.c
 * Abstract:
 *      An example C-file S-Functions for multiplying an input by 2,
 *      y  = 2*u
 * Real-Time Workshop note:
 *   This file can be used as is (noninlined) with the Real-Time
Workshop
 *   C rapid prototyping targets, or it can be inlined using the Target
 *   Language Compiler technology and used with any target. See
 *     matlabroot/toolbox/simulink/blocks/tlc_c/timestwo.tlc
 *     matlabroot/toolbox/simulink/blocks/tlc_ada/timestwo.tlc
 *   the C and Ada TLC code to inline the S-Functions.
*/

#define S_FUNCTION_NAME  timestwo
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"

/*================*
 * Build checking *
 *================*/
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, DYNAMICALLY_SIZED);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    if (!ssSetNumOutputPorts(S,1)) return;
    ssSetOutputPortWidth(S, 0, DYNAMICALLY_SIZED);
    ssSetNumSampleTimes(S, 1);
    /* Take care when specifying exception free code - see
sfuntmpl_doc.c */
    ssSetOptions(S,
                 SS_OPTION_WORKS_WITH_CODE_REUSE |
                 SS_OPTION_EXCEPTION_FREE_CODE |
                 SS_OPTION_USE_TLC_WITH_ACCELERATOR);
}
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
    ssSetModelReferenceSampleTimeDefaultInheritance(S);
}
static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T             i;
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T            *y    = ssGetOutputPortRealSignal(S,0);
    int_T             width = ssGetOutputPortWidth(S,0);
    for (i=0; i<width; i++) {
        /*
         * This example does not implement complex signal handling.
         * To find out see an example about how to handle complex signal
in
         * S-Functions, see sdotproduct.c for details.
```

```
        */
        *y++ = 2.0 *(*uPtrs[i]);
    }
}

static void mdlTerminate(SimStruct *S)
{
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file?
*/
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

11.8.3 连续状态方程

系统模型如图 11.27 所示，调用函数 csfunc.c 为 MATLAB 库自带实例。

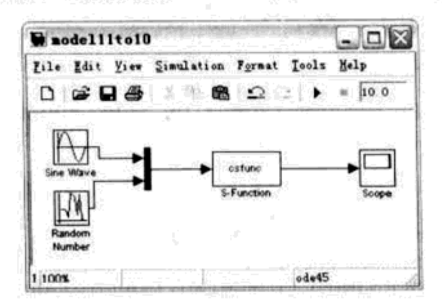


图 11.27　由 S 函数编写的连续系统

文件 csfunc.c 保存在 matlab/simulink/src/中，内容如下：

```
/*  File    : csfunc.c
 *  Abstract:
 *      Example C-file S-Functions for defining a continuous system.
 *      x' = Ax + Bu
 *      y  = Cx + Du
 *      For more details about S-Functions, see
simulink/src/sfuntmpl_doc.c.
*/

#define S_FUNCTION_NAME csfunc
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define U(element) (*uPtrs[element])  /* Pointer to Input Port0 */

static real_T A[2][2]={ { -0.09, -0.01 } , { 1   , 0   } };
static real_T B[2][2]={ { 1    , -7    } , { 0   , -2  } };
static real_T C[2][2]={ { 0    , 2     } , { 1   , -5  } };
static real_T D[2][2]={ { -3   , 0     } , { 1   , 0   } };

/*====================*
 * S-Functions methods *
 *====================*/

static void mdlInitializeSizes(SimStruct *S)
```

```
{
   ssSetNumSFcnParams(S, 0);  /* Number of expected parameters */
   if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
      return; /* Parameter mismatch will be reported by Simulink */
   }
   ssSetNumContStates(S, 2);
   ssSetNumDiscStates(S, 0);

   if (!ssSetNumInputPorts(S, 1)) return;
   ssSetInputPortWidth(S, 0, 2);
   ssSetInputPortDirectFeedThrough(S, 0, 1);

   if (!ssSetNumOutputPorts(S, 1)) return;
   ssSetOutputPortWidth(S, 0, 2);

   ssSetNumSampleTimes(S, 1);
   ssSetNumRWork(S, 0);
   ssSetNumIWork(S, 0);
   ssSetNumPWork(S, 0);
   ssSetNumModes(S, 0);
   ssSetNumNonsampledZCs(S, 0);

   /* Take care when specifying exception free code - see
sfuntmpl_doc.c */
   ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
   ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
   ssSetOffsetTime(S, 0, 0.0);
   ssSetModelReferenceSampleTimeDefaultInheritance(S);
}
#define MDL_INITIALIZE_CONDITIONS

static void mdlInitializeConditions(SimStruct *S)
{
   real_T *x0 = ssGetContStates(S);
   int_T  lp;

   for (lp=0;lp<2;lp++) {
      *x0++=0.0;
   }
}

/* Function: mdlOutputs =====================================
 * Abstract:
 *     y = Cx + Du
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
   real_T            *y    = ssGetOutputPortRealSignal(S,0);
   real_T            *x    = ssGetContStates(S);
   InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

   UNUSED_ARG(tid); /* not used in single tasking mode */
   /* y=Cx+Du */
   y[0]=C[0][0]*x[0]+C[0][1]*x[1]+D[0][0]*U(0)+D[0][1]*U(1);
   y[1]=C[1][0]*x[0]+C[1][1]*x[1]+D[1][0]*U(0)+D[1][1]*U(1);
}
```

```
#define MDL_DERIVATIVES

static void mdlDerivatives(SimStruct *S)
{
   real_T            *dx   = ssGetdX(S);
   real_T            *x    = ssGetContStates(S);
   InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

   /* xdot=Ax+Bu */
   dx[0]=A[0][0]*x[0]+A[0][1]*x[1]+B[0][0]*U(0)+B[0][1]*U(1);
   dx[1]=A[1][0]*x[0]+A[1][1]*x[1]+B[1][0]*U(0)+B[1][1]*U(1);
}

/* Function: mdlTerminate =====================================
 * Abstract:
 *    No termination needed, but we are required to have this routine.
 */
static void mdlTerminate(SimStruct *S)
{
   UNUSED_ARG(S); /* unused input argument */
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file?
*/
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

11.8.4 离散状态方程

系统模型如图 11.28 所示，调用函数 csfunc.c 为 MATLAB 库自带实例。

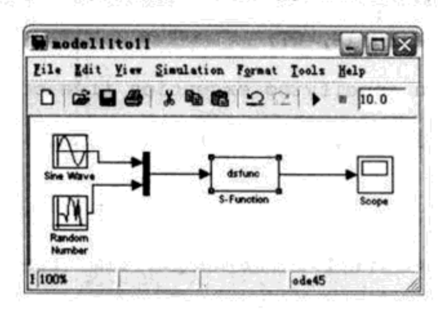


图 11.28 由 S 函数编写的离散系统

S 函数文件名为 dsfunc.c，文件保存在 matlab/simulink/src/中，内容如下：

```
/*  File    : dsfunc.c
 *  Abstract:
 *      Example C-file S-Functions for defining a discrete system.
 *      x(n+1) = Ax(n) + Bu(n)
 *      y(n)   = Cx(n) + Du(n)
 *      For more details about S-Functions, see
simulink/src/sfuntmpl_doc.c.
 */

#define S_FUNCTION_NAME dsfunc
#define S_FUNCTION_LEVEL 2
```

```
#include "simstruc.h"
#define U(element) (*uPtrs[element])  /* Pointer to Input Port0 */

static real_T A[2][2]={ { -1.3839, -0.5097 } , { 1      ,  0      }};
static real_T B[2][2]={ { -2.5559,  0      } , { 0      ,  4.2382 }};
static real_T C[2][2]={ {  0     ,  2.0761 } , { 0      ,  7.7891 }};
static real_T D[2][2]={ { -0.8141, -2.9334 } , { 1.2426,  0      }};

/*====================*
 * S-Functions methods *
 *====================*/

static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);  /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 2);

    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 2);
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 2);

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    /* Take care when specifying exception free code - see
sfuntmpl_doc.c */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}

/* Function: mdlInitializeSampleTimes ========================
 * Abstract:
 *    Specifiy that we inherit our sample time from the driving block.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, 1.0);
    ssSetOffsetTime(S, 0, 0.0);
    ssSetModelReferenceSampleTimeDefaultInheritance(S);
}

#define MDL_INITIALIZE_CONDITIONS

static void mdlInitializeConditions(SimStruct *S)
{
    real_T *x0 = ssGetRealDiscStates(S);
    int_T  lp;

    for (lp=0;lp<2;lp++) {
        *x0++=1.0;
```

```
    }
}

/* Function: mdlOutputs ======================================
 * Abstract:
 *      y = Cx + Du
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T             *y    = ssGetOutputPortRealSignal(S,0);
    real_T             *x    = ssGetRealDiscStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

    UNUSED_ARG(tid); /* not used in single tasking mode */
    /* y=Cx+Du */
    y[0]=C[0][0]*x[0]+C[0][1]*x[1]+D[0][0]*U(0)+D[0][1]*U(1);
    y[1]=C[1][0]*x[0]+C[1][1]*x[1]+D[1][0]*U(0)+D[1][1]*U(1);
}

#define MDL_UPDATE
/* Function: mdlUpdate ======================================
 * Abstract:
 *      xdot = Ax + Bu
 */
static void mdlUpdate(SimStruct *S, int_T tid)
{
    real_T             tempX[2] = {0.0, 0.0};
    real_T             *x       = ssGetRealDiscStates(S);
    InputRealPtrsType uPtrs    = ssGetInputPortRealSignalPtrs(S,0);

    UNUSED_ARG(tid); /* not used in single tasking mode */

    /* xdot=Ax+Bu */
    tempX[0]=A[0][0]*x[0]+A[0][1]*x[1]+B[0][0]*U(0)+B[0][1]*U(1);
    tempX[1]=A[1][0]*x[0]+A[1][1]*x[1]+B[1][0]*U(0)+B[1][1]*U(1);

    x[0]=tempX[0];
    x[1]=tempX[1];
}

/* Function: mdlTerminate ======================================
 * Abstract:
 *    No termination needed, but we are required to have this routine.
 */
static void mdlTerminate(SimStruct *S)
{
    UNUSED_ARG(S); /* unused input argument */
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file?
*/
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

11.8.5 混合系统

系统模型如图 11.29 所示，调用函数 csfunc.c 为 MATLAB 库自带实例。

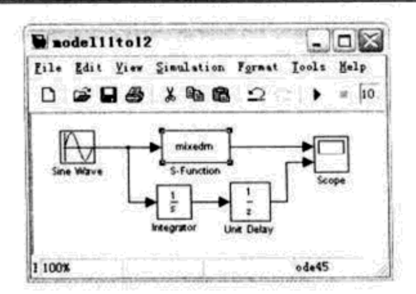


图 11.29　混合系统模型

混合系统 S 函数文件名为 mixedm.c，保存在文件夹 matlab/simulink/src/中，内容如下：

```
/*  File    : mixedm.c
 *  Abstract
 *     An example S-Functions illustrating multiple sample times by
implementing
 *        integrator -> ZOH(Ts=1second) -> UnitDelay(Ts=1second)
 *     with an initial condition of 1.
 *  (e.g. an integrator followed by unit delay operation).
 *     For more details about S-Functions, see
simulink/src/sfuntmpl_doc.c
*/
#define S_FUNCTION_NAME mixedm
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define U(element) (*uPtrs[element])  /* Pointer to Input Port0 */

/*====================*
 * S-Functions methods *
 *====================*/
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);  /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
       return; /* Parameter mismatch will be reported by Simulink */
    }
    ssSetNumContStates(S, 1);
    ssSetNumDiscStates(S, 1);
    ssSetNumRWork(S, 1);  /* for zoh output feeding the delay operator */
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 1);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetInputPortOffsetTime(S, 0, 0.0);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);
    ssSetOutputPortSampleTime(S, 0, 1.0);
    ssSetOutputPortOffsetTime(S, 0, 0.0);
    ssSetNumSampleTimes(S, 2);
    /* Take care when specifying exception free code - see
 sfuntmpl_doc.c. */
    ssSetOptions(S, (SS_OPTION_EXCEPTION_FREE_CODE |
                     SS_OPTION_PORT_SAMPLE_TIMES_ASSIGNED));
 } /* end mdlInitializeSizes */
 static void mdlInitializeSampleTimes(SimStruct *S)
 {
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
```

```
    ssSetSampleTime(S, 1, 1.0);
    ssSetOffsetTime(S, 1, 0.0);
    ssSetModelReferenceSampleTimeDefaultInheritance(S);
} /* end mdlInitializeSampleTimes */
#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S)
{
    real_T *xC0 = ssGetContStates(S);
    real_T *xD0 = ssGetRealDiscStates(S);
    xC0[0] = 1.0;
    xD0[0] = 1.0;
} /* end mdlInitializeConditions */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    /* update the internal "zoh" output */
    if (ssIsContinuousTask(S, tid)) {
        if (ssIsSpecialSampleHit(S, 1, 0, tid)) {
            real_T *zoh = ssGetRWork(S);
            real_T *xC  = ssGetContStates(S);
            *zoh = *xC;
        }
    }
    /* y=xD */
    if (ssIsSampleHit(S, 1, tid)) {
        real_T *y   = ssGetOutputPortRealSignal(S,0);
        real_T *xD  = ssGetRealDiscStates(S);
        y[0]=xD[0];
    }
} /* end mdlOutputs */

#define MDL_UPDATE
static void mdlUpdate(SimStruct *S, int_T tid)
{
    UNUSED_ARG(tid); /* not used in single tasking mode */
    /* xD=xC */
    if (ssIsSampleHit(S, 1, tid)) {
        real_T *xD = ssGetRealDiscStates(S);
        real_T *zoh = ssGetRWork(S);
        xD[0]=*zoh;
    }
} /* end mdlUpdate */

#define MDL_DERIVATIVES
static void mdlDerivatives(SimStruct *S)
{
    real_T            *dx   = ssGetdX(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    /* xdot=U */
    dx[0]=U(0);
} /* end mdlDerivatives */
static void mdlTerminate(SimStruct *S)
{
    UNUSED_ARG(S); /* unused input argument */
}
#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file?
*/
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

第12章　回 调 函 数

通过定义 MATLAB 表达式，当用户定义的模型的图表或者模块发生某种特殊的行为时被调用，如执行模块或者打开模块，这些表达式就称为回调函数。回调函数与模块、端口或者模型参数相关联。例如，双击模块时，会弹出此模块的对话框，对正弦信号模块，双击就会执行一个显示相应参数对话框的回调函数。其实，回调函数在其他编程语言中同样存在，类似于很多高级编程语言中的事件处理程序，如 VB、VC 等。

回调函数通常与 MATLAB 的图形处理有着密切的联系，使用图形工具创建了菜单之后，将菜单的每一个选项与相应的回调函数关联起来，当选项被选中时自动被执行。

本章主要内容包括：

- 回调函数基础
- 使用回调函数
- 模型结构命令
- 深入理解回调函数
- 回调函数实例
- 基于回调的图形用户界面

12.1　回调函数基础

模型的回调函数为交互式的或程序化的。使用 Model Properties 对话框的 Callbacks 选项卡创建交互式回调函数。使用 set_param 命令来执行对应的模型回调参数，创建一个程序化的回调函数。表 12.1 列出了模型回调参数，表 12.2 列出了模块回调函数。

表 12.1　模型回调参数

参数名	功　能
CloseFcn	用于模型关闭前，设置模型关闭的响应事件
PostLoadFcn	用于模型加载后，设置模型加载后的响应事件
InitFcn	用于模型开始仿真时，设置模型仿真开始的响应事件
PostSaveFcn	用于模型保存后，设置模型保存后响应事件
PreLoadFcn	用于模型加载前，设置模型加载前的响应事件
PreSaveFcn	用于模型保存前，设置模型保存前响应事件
StartFcn	用于模型仿真开始前，设置模型仿真开始前得响应事件
StopFcn	用于模型仿真结束后，设置模型仿真结束后的响应事件

表 12.2　模块回调函数

参数名	功　能
ClipboardFcn	用于模块被复制或者剪切到剪贴板时
CloseFcn	用于模块由 close_system 命令关闭时
CopyFcn	用于模块被复制时
DeleteChildFcn	用于删除子系统中的模块时
DeleteFcn	用于模块被删除时，主要关闭所有与此模型相关的图形窗口，此回调适用于子系统
DestroyFcn	用于模块被损坏时
InitFcn	用于模块编译及赋值前，此回调用来对模块进行初始化
LoadFcn	用于模块被加载后，适用于子系统。
ModelCloseFcn	用于模模型关闭前，设置响应事件，此回调适用于子系统
MoveFcn	当模块被移动或者被改变大小时
NameChangeFcn	用于模块名被修改时，或者模块路径被修改时
OpenFcn	用于模块打开时
ParentCloseFcn	用于包含当前模块的子系统关闭前
PreSaveFcn	用于模块保存前，此回调适用于子系统
PostSaveFcn	用于模块保存后，此回调适用于子系统
StartFcn	用于模块编译后与仿真开始前
StopFcn	用于仿真以任何形式关闭
UndoDeleteFcn	用于撤销模块删除操作

12.2　使用回调函数

使用 set_param 命令来加载回调函数，设定指定参数为指定值，使用语法为：

```
set_param('obj', 'parameter1', value1, 'parameter2', value2, ...)
```

其中，obj 为模型名或者模块路径的 MATLAB 字符串或 0，通常用 0 来设置参数的默认值。

例如，如果要调用模型为 vdp.mdl，则 obj 为‘vdp’，如果要调用模型 vdp.mdl 中的模块 Mu，这 obj 为“vdp/ Mu”。

parameter1 与 parameter2 为包含回调参数的 MATLAB 字符串。

value1 与 value2 分别为与 parameter1 与 parameter2 参数相对应的参数值。

【实例 1201】设置 vdp 模型的 Solver(求解器)与 StopTime(终止时间)参数。

```
>> set_param('vdp', 'Solver', 'ode15s', 'StopTime', '3000')
```

【实例 1202】设置 vdp 模型中的模块 Mu 的 Gain(增益)参数为 1000(刚性方程)。

```
>> set_param('vdp/Mu', 'Gain', '1000')
```

【实例 1203】设置 vdp 模型中 Fcn 模块的形状，结果对比如图 12.1 和图 12.2 所示。

```
>> set_param('vdp/Fcn', 'Position', [50 100 110 120])
```

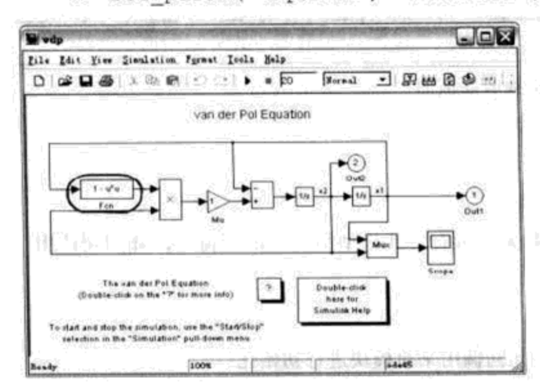


图 12.1　修改形状前

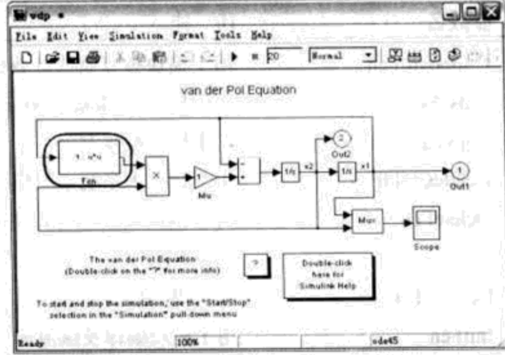


图 12.2　修改形状后

【实例 1204】设置 mymodel 模型中 Zero-Pole(零极点)模块的 Zeros 与 Poles 参数。

```
>> set_param('mymodel/Zero-Pole','Zeros','[2 4]','Poles','[1 2 3]')
```

【实例 1205】设置封装子系统中模块的 Gain 参数，变量 k 与这个 Gain 参数关联。

```
>> set_param('mymodel/Subsystem', 'k', '10')
```

【实例 1206】设置 mymodel 模型中的 Compute 模块的 OpenFcn 回调参数，但用户双击 Compute 模块时执行函数'my_open_fcn'。

```
>> set_param('mymodel/Compute', 'OpenFcn', 'my_open_fcn')
```

12.3　模型结构命令

MATLAB 提供了许多用来生成、编辑 Simulink 模型的命令，可以生成、保存、增加或者删除模块或者连线、获取或者设置模型及模块的参数。其中与回调函数有关的命令如表 12.3 所示。

表 12.3　常用的与回调函数有关的命令

命令形式	功　能
bdroot(object)	返回当前模型的模块名
gcb	返回当前模块的全路径
gcs	返回当前系统或子系统的全路径
get_param(obj,param)	获取当前系统、子系统的参数值
set_param(obj,param,value)	设置当前系统、子系统的参数值

1. 返回模型的模型名

命令格式为：

```
bdroot
bdroot('obj')
```

【实例 1207】返回当前模型的模型名称。

```
>> bdroot(gcb)
```

2. 返回当前模块的路径

命令格式为：

```
>> gcb
gcb('sys')
```

其中：

gcb 返回当前系统的当前模块的完整路径。

gcb('sys') 返回指定系统模型的当前模块的完整路径。

【实例 1208】返回最近打开模型中被选择的模块的路径。

```
>> gcb
ans =
  clutch/Locked/Inertia
```

【实例 1209】返回当前模块的 Gain 参数值。

```
>> get_param(gcb,'Gain')
ans =
  1/(Iv+Ie)
```

3. 返回当前模型的名称

命令格式为：

```
gcs
```

【实例 1210】返回包含最近被选择模块系统的路径。

```
>> gcs
ans =
  clutch/Locked
```

4. 获取当前系统、子系统的参数值

语法格式为：

- get_param('obj', 'parameter')
- get_param({ objects }, 'parameter')
- get_param(handle, 'parameter')
- get_param(0, 'parameter')
- get_param('obj', 'ObjectParameters')
- get_param('obj', 'DialogParameters')

说明：

- get_param('obj', 'parameter')返回指定参数值，其中'obj'是系统或者模块的路径名。
- get_param({ objects }, 'parameter')返回多个参数值，这多个目标由 { objects }细胞矩阵定义。
- get_param(handle, 'parameter')返回系统的指定参数，系统由句柄决定。

- get_param(0, 'parameter') 返回 Simulink 任务参数的当前值，或者模型或者模块的默认值。
- get_param('obj', 'ObjectParameters') 返回一个结构体数据，用来描述 obj 系统的参数。结构体中的每一个域都对应一个特殊的参数，并包含参数名。例如，对应于 object 的 Name 参数的参数名域(Name 域)，每一个参数域本身又包含 3 个子域，分别为 Name，Type 和 Attributes，分别用来指定参数名(如，Gain)、数据类型(如字符串)和属性(如只读)。
- get_param('obj', 'DialogParameters') 返回一个细胞矩阵，包含指定模块参数对话框的参数名。

【实例 1211】返回系统 clutch 中子系统 Requisite Friction 中 Inertia 模块 Gain 参数值。

```
>> get_param('clutch/Requisite Friction/Inertia','Gain')
ans =
   1/(Iv+Ie)
```

【实例 1212】返回 vdp.mdl 模型中的所有模块的类型。

```
>> vdp
>> blks = find_system(gcs, 'Type', 'block');
>> listblks = get_param(blks, 'BlockType')
listblks =
    'Fcn'
    'SubSystem'
    'SubSystem'
    'Gain'
    'Mux'
    'Product'
    'Scope'
    'Sum'
    'Integrator'
    'Integrator'
    'Outport'
    'Outport'
```

【实例 1213】返回 vdp.mdl 模型中被选择模块 Fcn 的模块名。

```
>> get_param(gcb, 'Name')
ans =
Fcn
```

【实例 1214】返回当前 vdp.mdl 模型中所选择模块 Fcn 的属性。

```
>> p = get_param(gcb, 'ObjectParameters');
>> a = p.Name.Attributes
a =
    'read-write'    'dont-eval'    'always-save'
```

【实例 1215】返回 vdp.mdl 模型中 Mu 模块对话框的参数。

```
>> p = get_param('vdp/Mu', 'DialogParameters')
p =
                     Gain: [1x1 struct]
           Multiplication: [1x1 struct]
    ParameterDataTypeMode: [1x1 struct]
        ParameterDataType: [1x1 struct]
```

```
        ParameterScalingMode: [1x1 struct]
            ParameterScaling: [1x1 struct]
             OutDataTypeMode: [1x1 struct]
                 OutDataType: [1x1 struct]
                  OutScaling: [1x1 struct]
                   LockScale: [1x1 struct]
                     RndMeth: [1x1 struct]
SaturateOnIntegerOverflow: [1x1 struct]
                  SampleTime: [1x1 struct]
```

5. 设置当前系统、子系统的参数值

详细情况参见 12.2 节。

12.4 深入理解回调函数

前面讲了许多回调函数的各种命令，但是对回调函数的具体理解可能还有一些欠缺。下面从 MATLAB 自带的一个实例讲解回调函数的作用、具体设置以及能够完成的工作等。打开已存在模型 f14，在 MATLAB 命令窗口输入：

```
>> f14
```

f14 是一个复杂的 Simulink 模型，用户可以在 MATLAB demo 窗口找到这个模型，如图 12.3 所示。在打开模型 f14 时，用户可能会被复杂的内容所迷惑。这里不需要知道模型的具体内容，或者模型的构造技巧，主要体会一下回调函数的作用。

在 f14 模型中的模块中有很多变量，在运行模型之前，用户并没有人工地对模型中的变量赋值及定义。但是运行模型前必须在 MATLAB 工作空间定义这些变量，否则模型是不能运行的。但是打开 f14 模型时，并没有在 MATLAB 工作空间中定义这些变量，至少表面上没有进行这些操作，我们可以得到仿真结果，如图 12.4 所示。

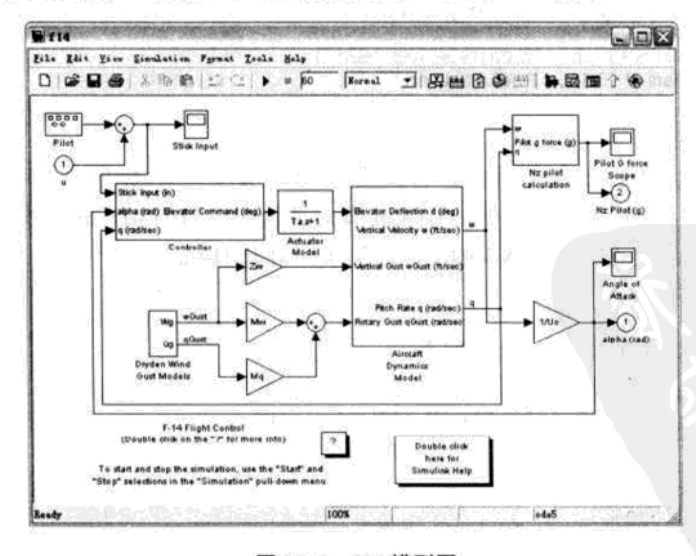


图 12.3 f14 模型图

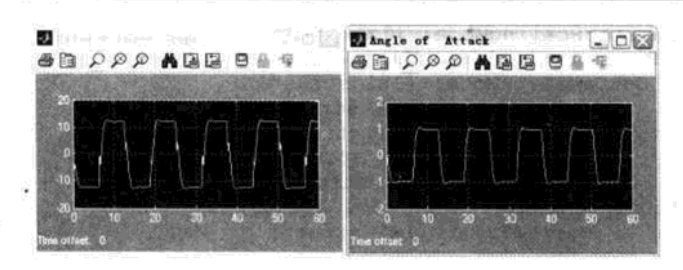


图 12.4 模型仿真结果

为确定是否在 MATLAB 工作空间定义了模型中的变量，在 MATLAB 命令窗口输入

```
>> who
Your variables are:
Beta     Kf     Md     Sa     Tal    Vto    Wa     a      g
Gamma    Ki     Mq     Swg    Ts     W1     Zd     b
Ka       Kq     Mw     Ta     Uo     W2     Zw     cmdgain
```

可以看出，在运行仿真之前，MATLAB 工作空间中已经定义了模型中所需要的变量，只是自动完成的而已。这种方式不同于手动输入，是 MATLAB 自动调用函数来完成的。一个基本的可能实现路径是，在模型被打开的同时调用了某个数据文件或者函数文件。这种实现的方式就是 f14 模型所使用的，这就是回调函数在其中起的作用。

f14 模型在载入之前就自动执行了对变量赋值的 M 文件。为模型设置参数的命令为 set_param，具体用法可参见 12.2 节。此处使用的是模型回调函数，而且是在模型载入之前，从表 12.1 中可知使用的回调函数应该是 PreLoadFcn。假设定义变量参数的数据文件为 modeldat.m。假定模型 exModel.mdl 中完成类似 f14 模型中定义变量并赋值的命令为：

```
>>set_param('exModel','PreLoadFcn','modeldat');
```

为了验证 f14 模型中是否是使用上述方法实现对变量赋值的，可以使用如下命令进行验证，结果表明，上述想法是完全正确的。

```
>> f14
>> get_param('f14','PreLoadFcn')
ans =
f14dat
```

从命令执行的结果可以看出，f14 模型中定义了一个数据文件为 f14dat.m，在 Simulink 载入 f14.mdl 之前，文件 f14dat.m 就会被执行。当 f14dat.m 被执行后，所定义和赋值的变量就会留存在 MATLAB 的工作空间中。打开 f14dat.m 文件：

```
>> edit f14dat
```

则会显示 f14dat 文件的内容，具体如下：

```
% Numerical data for F-14 demo
%  Copyright 1990-2002 The MathWorks, Inc.
%  $Revision: 1.16 $
g = 32.2;
Uo = 689.4000;
Vto = 690.4000;
% Stability derivatives
Mw = -0.00592;
Mq = -0.6571;
```

```
Md = -6.8847;
Zd = -63.9979;
Zw = -0.6385;
% Gains
cmdgain = 3.490954472728077e-02;
Ka = 0.6770;
Kq = 0.8156;
Kf = -1.7460;
Ki = -3.8640;
% Other constants
a = 2.5348;
Gamma = 0.0100;
b = 64.1300;
Beta = 426.4352;
Sa = 0.005236;
Swg = 3;
Ta = 0.0500;
Tal = 0.3959;
Ts = 0.1000;
W1 = 2.9710;
W2 = 4.1440;
Wa = 10;
```

f14.mdl 模型使用了如下回调函数：

```
>>set_param('f14','PreLoadFcn','f14dat');
```

用户可以自己测试一下。首先将 f14.mdl 复制一份为 f14copy.mdl，然后修改 f14datcopy.m，执行下列命令可以察看修改后模型的运行结果：

```
>> f14copy
```

12.5　回调函数实例

下面采用 6.2 节中的实例，这个实例模型在实际中是非常普遍的，如建筑结构的振动、多体动力学等，在此不讨论其实际意义。在 6.2 节中对模型中的参数赋值是通过手动执行的，没有赋值之前，模型是不能得到运行结果的。

本节建立与 6.2 节同样的模型，保存为 model12to01.mdl，如图 12.5 所示，其中 State-Space 模块的参数设置如图 12.6 所示。

模型中的 State-Space 模块中有 4 个变量需要赋值，分别为 A、B、C 和 D。如果在运行模型之前不对这 4 个变量赋值，则运行模型时就会出错。

设本实例的数据文件为 mfun12to01.m，内容如下：

```
% Numerical data for model12to01 demo
% Copyright SciEi.com and Dytrol.com.
m=1;
c=4;
k=3;
M=m*eye(3);
P=c*[2 -1 0;-1 2 -1;0 -1 1];
K=k*[2 -1 0;-1 2 -1;0 -1 1];
G=[0;0;1];
A=cat(1,cat(2,zeros(3,3),eye(3)),cat(2,-inv(M)*K,-inv(M)*P));
B=cat(1,zeros(3,1),-inv(M)*G);
C=eye(6);
```

```
D=zeros(6,1);
```

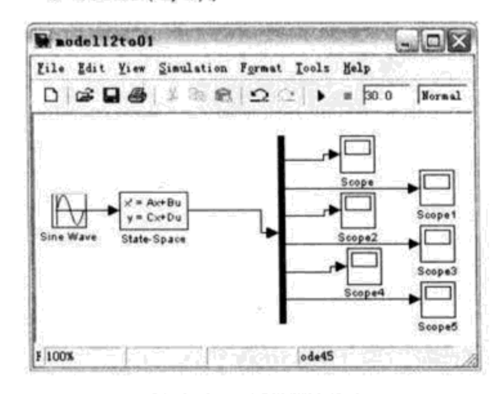


图 12.5 实例模型图

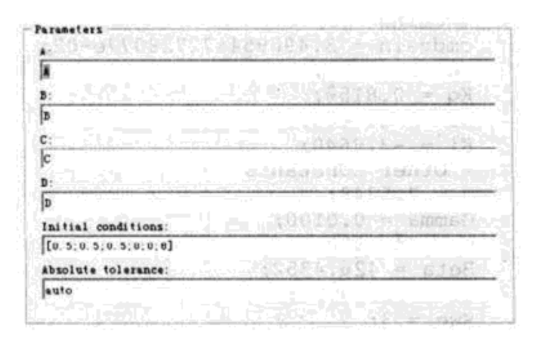


图 12.6 State-Space 模块的参数设置

建立模型文件 model12to01.mdl 与参数文件 mfun12to01.m 之后，就可以在 MATLAB 命令窗口执行回调命令了，输入：

```
>> clear
>> model12to01
>> set_param('model12to01','PreLoadFcn','mfun12to01');
```

保存模型文件 model12to01.mdl，再关闭模型文件，然后重新打开 model2to01.mdl 模型，此时在工作空间就会出现许多变量，可在 MATLAB 命令窗口输入：

```
>> who
Your variables are:
A  B  C  D  G  K  M  P  c  k  m
```

然后运行 model12to01.mdl 模型，可以看到模型能够正常运行。

12.6 基于回调的图形用户界面

12.6.1 图形用户界面设计的基本原则

如果没有一定的设计原则，就会因为要求的不同而设计出千差万别的界面。为了满足使用和一定的审美观，一般应该遵循 3 个原则：简单性、一致性和习惯性。

1. 简单性原则

在开发者设计时，为了能够使得操作简单明了，应该力求界面简单、直接并清晰地体现出界面的功能。不要将可有可无的功能放到界面上。设计界面最好直观，多采用图形化界面。尽量减少窗口数量，免得操作时在多个窗口间切换。

2. 一致性原则

这主要是针对新手来说。主要包含两个方面：首先，每个开发者所开发的界面应该风格一致，每个界面的操作方法类似；其次，开发者所开发的界面应该与所开发的环境保持一致，可以让新手更容易地熟悉环境。

3. 习惯性原则

设计界面所使用的各种参数意义，各种参数符号以及标志都要与传统的、人们熟悉的一致，可以让新手直接通过这些熟悉的符号来了解参数的具体意义。

4. 其他多种因素

如保持界面的动态性，就能够及时地响应用户的操作，并能够将结果反映出来。对操作要迅速连续，对时间比较长的操作，尽量提供等待时间进程提示，使用户知道操作在继续中。

图形界面的设计步骤，简单归纳一下界面设计的基本步骤如下，在实际过程中，各个步骤可以穿插操作，在于个人的习惯。

(1) 分析要实现的功能，明确设计任务。

(2) 绘制草图，设计合理的结构。

(3) 在界面上静态实现草图中的结构。

(4) 编写动态程序，调试界面功能。

以上步骤可能需要反复调试，反复设计才能够实现。

12.6.2　建立动态对话框实例

这里只通过一个简单例子说明其中的使用方法。这个例子并不是一个完整的实例。只是完成一个特定的功能。

本实例的目的主要是为了让 Make the Gain disable 复选框和 Gain 复选框不能同时被选择，即完成单选框的功能。目的是在不打开模块的条件下，双击模块即可调换选项的选中状态。

首先拖放一个 Subsystem 模块到模型窗口，保存为 model12to02.mdl，设置子系统如图 12.7 所示。设置完成后，双击 Subsystem 模块弹出如图 12.8 所示参数对话框。为设置方便，用右键拖放来复制此模块，出现 Subsystem1 模块。使 Subsystem1 模块处于被选中状态。

在 MATLAB 命令窗口输入：

```
>> set_param(gcb,'OpenFcn','mfun12to02')
>> get_param(gcb,'MaskValues')
ans =
    'off'
    'on'
```

从获取的 MaskValues 值可以看出， Make the Gain disable 复选框是未选中的状态，而 Gain 复选框是选中的状态。然后双击 Subsystem1 模块，并保存。此时在 MATLAB 命令窗口输入：

```
>> get_param(gcb,'MaskValues')
ans =
    'on'
    'off'
```

可以看出，Make the Gain disable 复选框是被选状态，而 Gain 复选框为未选中状态。

说明双击模块使得两个选项相互调换。在此双击，可以得到下面的结果：

```
>> get_param(gcb,'MaskValues')
ans =
    'off'
    'on'
```

其中回调命令中使用的 mfun12to02.m 内容如下：

```
% mfun12to02.m
M_value=get_param(gcb,'MaskValues'); %获取两个复选框的状态
if strcmp(char(M_value(1)),'on')        %判断第一个状态是否被选中
  set_param(gcb,'MaskValues',{'off','on'});%第一个复选框未选,第二个复选为被选状态
else
  set_param(gcb,'MaskValues',{'on','off'});%第一个复选框被选,第二个复选为未选状态
end
```

至此，完成对一个简单的动态对话框回调函数的使用。实例虽然简单，但再复杂的模型，也是这样一步一步累积起来的。只要能够多练多用，便可熟能生巧。

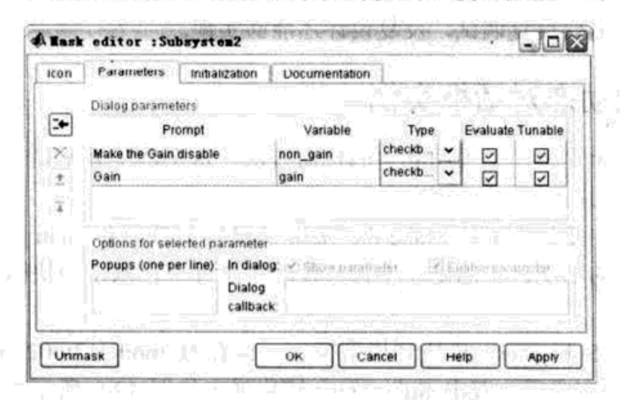


图 12.7　子系统封装对话框设置

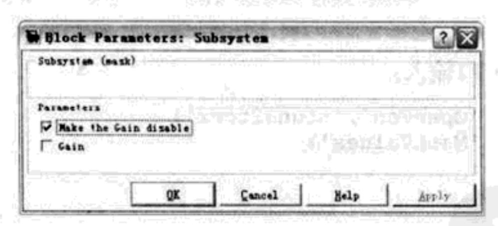


图 12.8　子系统动态对话框

第 13 章　图 形 动 画

在仿真过程中，总希望能够实现结果的动画输出，如单摆的摆动，结构的振动等，这样能够以直观的方式显示结果。这一章就介绍如何把图形动画添加到 Simulink 模型中。

S 函数的动画就是一个有没有状态函数、没有输出变量的 S 函数生成动画，所以只能作为一个常规的结果显示。S 函数主要由两个部分组成：初始化部分和更新部分。

- 在初始化过程中要创建图形窗口及动画对象。
- 更新过程中动画对象的属性作为 S 函数模块的输入函数，动画的变化是随动画对象的属性而变化。

本章主要内容包括：

- 动画显示的初始化
- 动画的更新
- 单摆动画显示实例

13.1　动画显示的初始化

S 函数动画的初始化语句包括 S 函数的初始化和图形初始化。应该设置较小的采样时间，以便动画能够较为连续地显示出来。理论上是采样数值越小越好，但是考虑到实际计算机系统，采样时间过小往往会使仿真运算很慢。对于连续系统，不要将采样时间设置为 0，会使动画显示不稳定。

在仿真开始时需要检测当前 S 函数是否已经打开动画图形，相应的 MATLAB 程序格式为如下形式：

```
if(findobj('UserData',gcb))
%若模型已经打开，则空操作
else
%添加图形初始化操作
end
```

其中的初始化图形语句用 figure 命令，语法格式如下：

```
h_fig=figure('Position',[x_pos,y_pos,width,height]);
```

然后，将当前 S 函数模块的路径保存到图形 UserData 中，设置格式为：

```
set(hfig,'UserData',gcb);
```

绘制图动画可用 MATLAB 的 plot 命令，如果要绘制向量 Xarray 和向量 Yarray 定义的曲线，具体格式如下：

```
handl=plot(Xarray, Yarray);
```

绘制好图形之后，需要将图形的句柄进行保存。假设保存绘制两个图形的动画，句柄分别为 hand1 与 hand2。下面的语句会把它们保存到一个结构中：

```
Tdata.hand1=hand1;
Tdata.hand2=hand2;
```

13.2 动画的更新

S 函数是以一个离散的模式进行采样的，所以可以看作是一个离散模型。flag=2 表示在采样时刻执行 S 函数。

更新函数会从 S 函数的 UserDate 变量中获取改变图形的句柄，例如：如果句柄以结构变量的形式储存，有如下形式：

```
Tdata=get_param(gcb,'UserData');
hand1=Tdata.hand1;
hand2=Tdata.hand2;
```

然后计算改变图形对象的属性值，然后使用 set 函数来更新属性：

```
set(handle.propertyName,propertyValue);
```

其中 handle 为对象的句柄，propertyName 是将要被修改属性的对象名的 MATLAB 字符串，propertyValue 为新的属性值。

13.3 单摆动画显示实例

建立模型如图 13.1 所示，在此只说明动画 S 函数的实现方式。其中 Constant 模块与 Sine Wave 模块均为相应的默认值，前者为单摆的支点，后者为单摆的摆动点。Animation Function1 模块的参数设置如图 13.2 所示，S-Function Name 参数设置为 aopndanim1，S-Function Parameters 设置为 0.05。结果输出如图 13.3 所示。

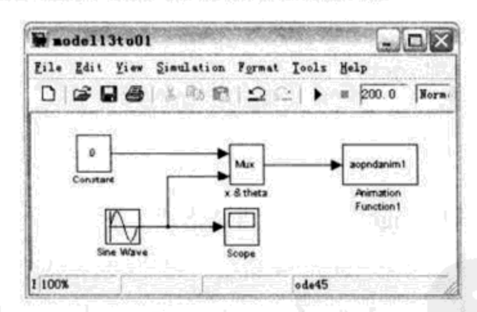


图 13.1 单摆动画显示实例模型图

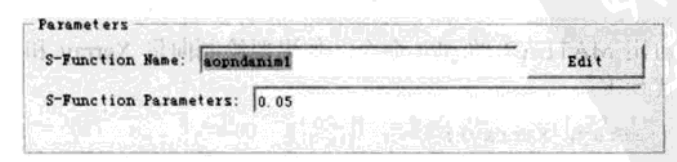


图 13.2 Animation Function1 模块的参数设置



图 13.3　模型动画输出结果界面

其中 aopndanim.m 函数的内容如下：

```
function [sys,x0]=aopndaniml(t,x,u,flag,ts);
%PNDANIM3 S-function for animating the motion of a pendulum.
global PendAniml
if flag==2,
  if any(get(0,'Children')==PendAniml),
    if strcmp(get(PendAniml,'Name'),'simppend Animation'),
      set(0,'currentfigure',PendAniml);
      hndlList=get(gca,'UserData');
      x=[u(1) u(1)+2*sin(u(2))];
      y=[0 -2*cos(u(2))];
      set(hndlList(1),'XData',x,'YData',y);
      set(hndlList(2),'XData',u(1),'YData',0);
      drawnow;
    end
  end
  sys=[];
elseif flag == 4 % Return next sample hit
  % ns stores the number of samples
  ns = t/ts;
  % This is the time of the next sample hit.
  sys = (1 + floor(ns + 1e-13*(1+ns)))*ts;
elseif flag==0,
  % Initialize the figure for use with this simulation
  animinit('simppend Animation');
  [flag,PendAniml] = figflag('simppend Animation');
  axis([-3 3 -2 2]);
  hold on;
  x=[0 0];
  y=[0 -2];
  hndlList(1)=plot(x,y,'LineWidth',5,'EraseMode','background');
  hndlList(2)=plot(0,0,'.','MarkerSize',25,'EraseMode','back');
  set(gca,'DataAspectRatio',[1 1 1]);
  set(gca,'UserData',hndlList);
  sys=[0 0 0 2 0 0];
  x0=[];
end
```

第 14 章　SimPowerSystems 在电路仿真中的应用

要掌握 SimPowerSystems 工具箱，必须了解电子电路的建模与仿真。本章首先介绍 SimPowerSystems 工具箱的模块库以及仿真命令，这是利用 SimPowerSystems 的基础。在此基础上，通过对一个较为复杂的实例进行讲解来深入理解 SimPowerSystems 的强大功能和作用。SimPowerSystems 可以对电路系统、电力电子系统、电机系统、电力传输过程等进行仿真。

本章主要内容包括：

- SimPowerSystems 模块库
- 模拟电路仿真实例

14.1　SimPowerSystems 模块库

通过模块库浏览器打开或者直接在 MATLAB 命令窗口输入：

```
>> powerlib
```

就会弹出如图 14.1 所示的 SimPowerSystems(电力系统)模块库窗口。



图 14.1　SimPowerSystems 模块库

在 SimPowerSystems 模块库中有 9 个模块集，每个模块集中又包含了若干模块，双击每个模块集就能够看到相应模块集中的模块，分别介绍如下：

1. Electrical Sources(电源模块集)

双击 Electrical Sources 模块集，弹出窗口如图 14.2 所示，包括 DC Voltage Source(直流电压源)、AC Voltage Source(交流电压源)、AC Current Source(交流电流源)、Controlled Voltage Source(可控电压源)、Controlled Current Source(可控电流源)、Three-Phase Source(三相电源)和 Three-Phase Programmable Source(三相程控电源)7 个模块。

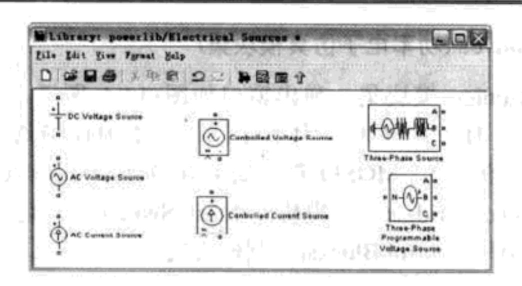


图 14.2　电源模块集

2. Elements(电路元件子模块集)

双击 Elements 模块集，弹出如图 14.3 所示的窗口，包括 Series RLC Branch(串联 RLC 分支)、Series RLC Load(串联 RLC 负载)、Parallel RLC Branch(并联 RLC 分支)、Parallel RLC Load(并联 RLC 负载)、Three-Phase Series RLC Load(三相串联 RLC 负载)、Three-Phase Parallel RLC Load(三相并联 RLC 负载)、Three-Phase Harmonic Filter(三相谐波滤波器)、Mutual Inductance(互感)、Three-Phase Mutual Inductance(三相互感)、Three-Phase Dynamic Load(三相动力负载)、Ground(接地)、Neutral Node 10(中性节点)、Connection Port(连接端口)、Pi Section Line(π 界面导线)、Distributed Parameters Line(分布参数导线)、Three-Phase PI Section Line(三相 π 界面导线)、Breaker(断路器)、Three-Phase Breaker(三相断路器)、Three-Phase Fault(三相断路故障)、Linear Transformer(线性转换器)、Saturable Transformer(饱和转换器)、Multi-Winding Transformer(多线圈转换器)、Three-Phase Transformer (Two Windings)(两线圈三相转换器)、Three-Phase Transformer (Three Windings)(三线圈三相转换器)、Zigzag Phase-Shifting Transformer(锯齿相位滑移转换器)和 Three-Phase Transformer 12 Terminals(12 终端三相转换器)。

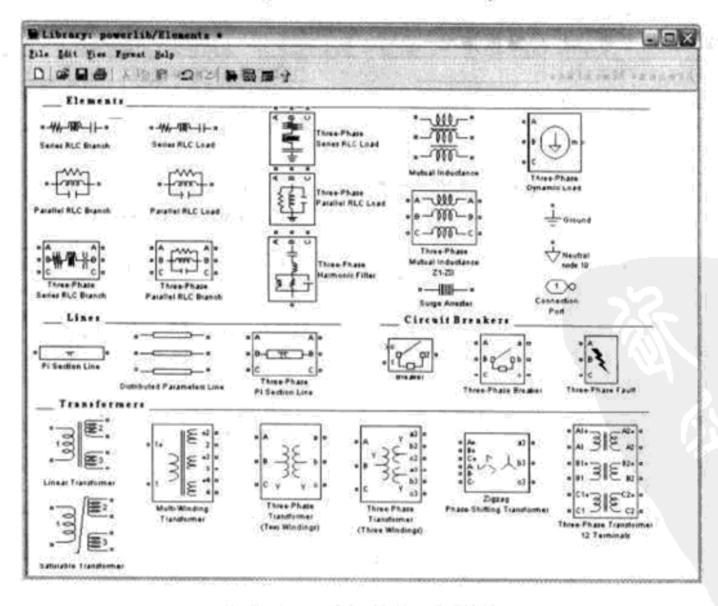


图 14.3　电路元件子模块集

3. Power Electronics(功率电子仿真模块集)

双击 Power Electronics 模块集，弹出窗口如图 14.4 所示，包括 Diode(二极管)、Gto(可关断可控硅)、Thyristor(半导体闸流管)、IGBT(绝缘栅二极管)、Detailed Thyristor(精细半导体闸流管)、MOSFET(金属氧化物半导体场效应晶体管)、Universal Bridge(电桥)、Three-Level Bridge(三级电桥)、Ideal Switch(理想切换器)、Discrete Control Blocks(离散控制模块集)和 Control Blocks(控制模块集)。

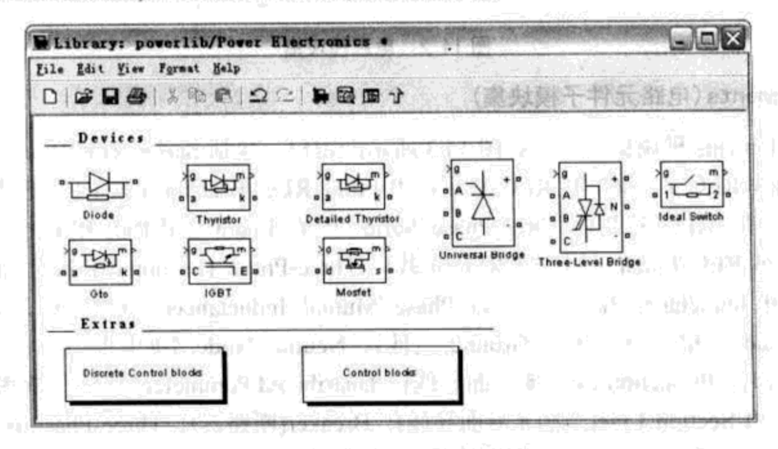


图 14.4 功率电子仿真模块集

4. Machines(机构模块集)

双击 Machines 模块集，弹出窗口如图 14.5 所示。

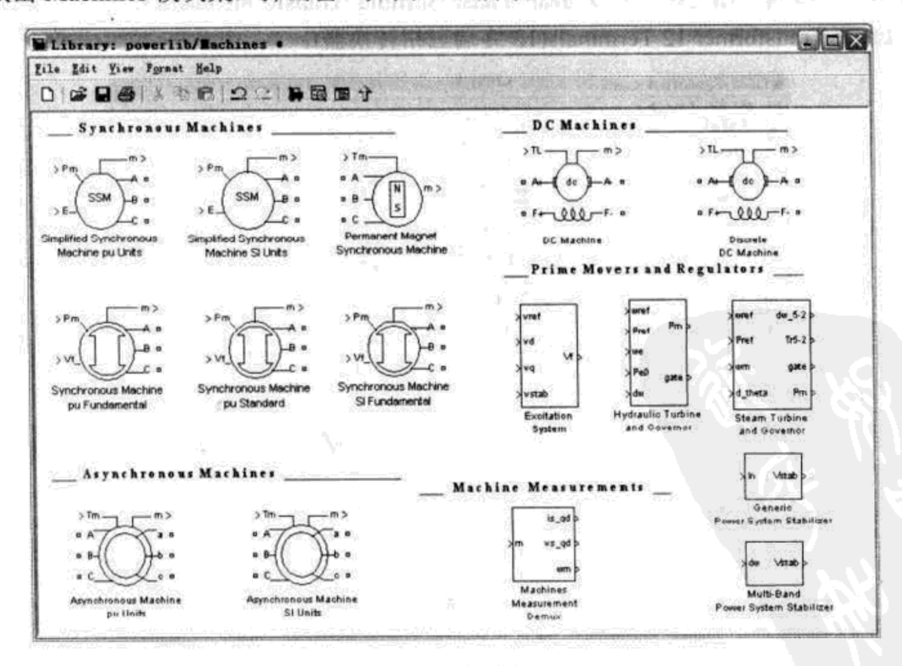


图 14.5 机构模块集

- Synchronous Machine 类：异步电机类，对三相凸极点异步电机进行建模，包括：Simplified Synchronous Machine pu Units、Simplified Synchronous Machine SI Units、Permanent Magnet Synchronous Machine、Synchronous Machine pu Fundamental、Synchronous Machine pu Standard 和 Synchronous Machine SI Fundamental 模块。
- DC Machine 类：直流电机类，对一个独立的被激励的直流机构进行建模，包括 DC Machine 和 Discrete DC Machine 模块。
- Asynchronous Machine 类：异步电机类，对一个三相异步电机动力学系统进行建模，包含 Asynchronous Machine pu Units 和 Asynchronous Machine SI Units 模块。
- Machine Measurement 类：机器测量类，将一个信号分成多个信号，包含模块为 Machine Measurement Demux。

5. Measurements(测量模块集)

双击 Measurements 模块集，弹出窗口如图 14.6 所示，包括：Current Measurement(电流测量)、Voltage Measurement(电压测量)、Impedance Measurement(阻抗测量)、Multimeter(万用表)、Three-Phase V-I Measurement(三相图间距测量)，以及 Continuous Measurement(连续测量)、Discrete Measurement(离散测量)和 Phasor Measurement(相图测量)模块。

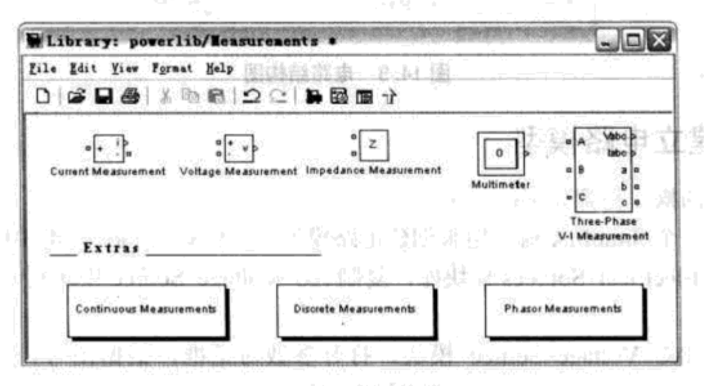


图 14.6　测量模块集

6. Phasor Elements(相图元件模块集)

双击 Phasor Elements 模块集，弹出窗口如图 14.7 所示，只有 Static Var Compensator(静态无功补偿)模块。

7. 其他模块集

包括 Extras、Powergui 和 Demos 模块集就不在此详细介绍了。

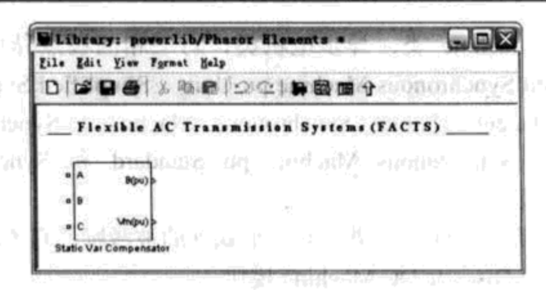


图 14.7 相图元件模块集

14.2 模拟电路仿真实例

下面通过实例详细介绍模拟电路的仿真模块和仿真命令的使用方法，并介绍如何将 14.1 节中介绍的模块进行有机的结合。

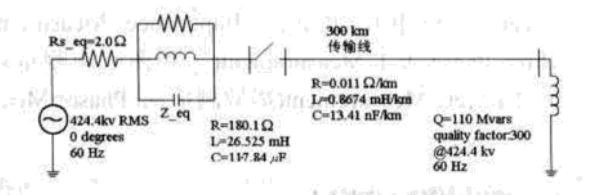


图 14.8 电路结构图

14.2.1 建立电路模型

建立模型的操作步骤如下：

(1) 新建一个 Simulink 窗口用来创建电路模型，并保存为 model14to01.mdl。

(2) 打开 Electrical Sources 模块库，复制 AC Voltage Source 模块到 model14to01.mdl 模型窗口。

(3) 双击 AC Voltage Source 模块，打开参数对话框，根据图 14.8 中的参数设置 amplitude, Phaset 和 Frequency 参数，如图 14.9 所示。

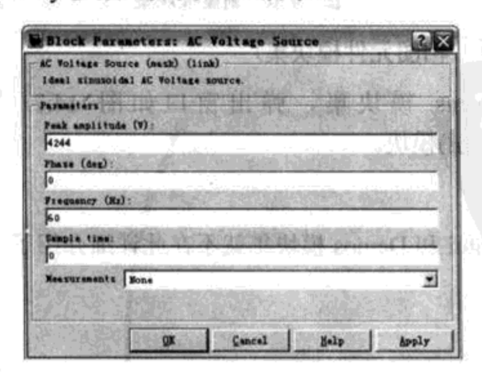


图 14.9 AC Voltage Source 模块参数对话框

(4) 将模块名 Voltage Source 改为 Vs。

(5) 从 Elements 模块库中复制 Parallel RLC Branch 模块到模型窗口中，并双击模块，根据图 14.8 设置对话框参数，如图 14.10 所示，并将模块重新命名为 Z_eq。

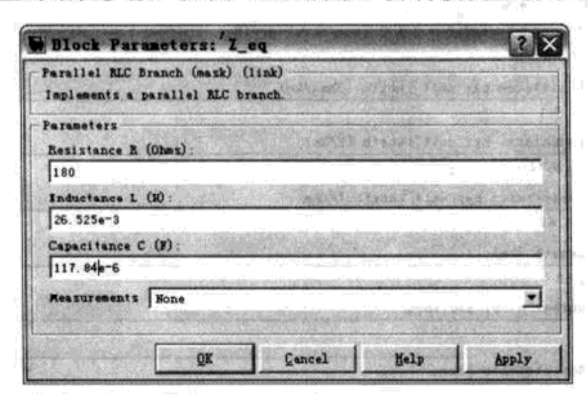


图 14.10　Parallel RLC Branch 参数对话框

(6) 图 14.8 中描述的电阻 Rs_eq 可以直接通过 Parallel RLC Branch 模块来实现。可以直接右键拖放复制窗口中已经存在的 Parallel RLC Branch 模块，命名为 Rs_eq，设置参数 Resistance R 为 2.0，Inductance L 为 inf，Capacitance C 为 0。

注意： 一旦参数设置好之后，可以看到窗口中的 Rs_eq 模块的 L 和 C 部分就会自动消失，此时就变成一个单纯的电阻了。还可以通过 Series RLC Branch 模块来得到同样的效果，只要分别设置 L 和 C 为 0 和 inf。

(7) 从 Elements 模块组中复制 Ground 模块到模型窗口。

(8) 适当地设置模块的大小，并用连线将模块连接起来，得到如图 14.11 所示模型。

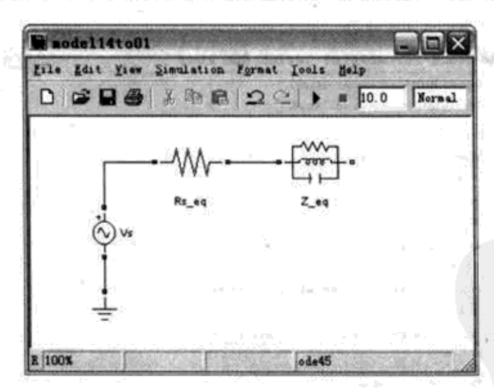


图 14.11　未完成模型图

(9) 从 Elements 模块库中复制 PI Section Line 模块到 model14to01.mdl 模型中，双击模块，按照图 14.8 设置对话框，如图 14.12 所示。

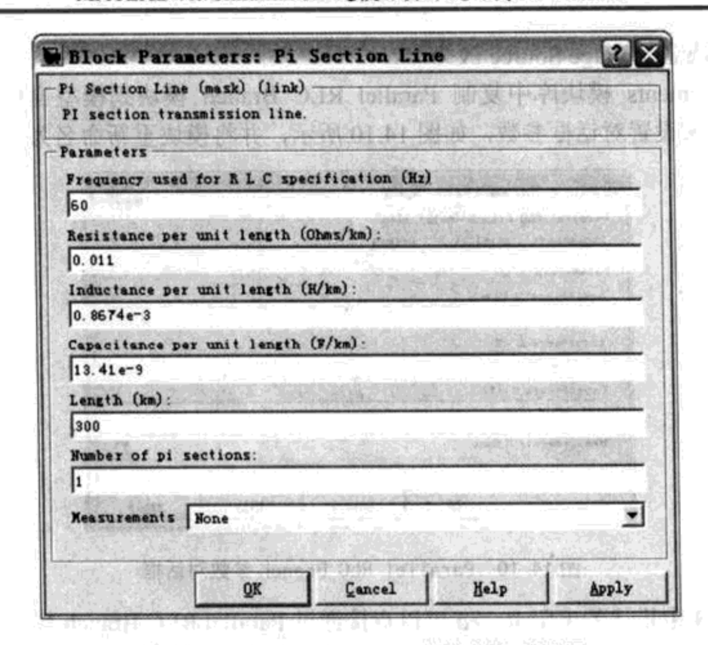


图 14.12 PI Section Line 模块参数对话框

(10) 从 Elements 模块库中复制 Series RLC Load 模块到 model14to01.mdl 模型中，并重命名为 110 Mvar，如图 14.13 所示。设置参数如下：

- Vn：424.4e3 V
- fn：60 Hz
- P：110e6/300 W (quality factor = 300)
- QL：110e6
- Qc：0

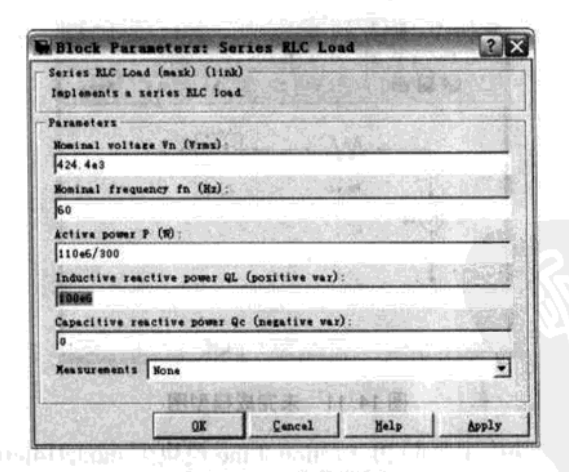


图 14.13 参数设置

(11) 从 Elements 模块组中复制 Ground 模块到模型窗口，或者直接复制模型中已存在的 Ground 模块。

(12) 连接剩余的模块，最后得到模型如图 14.14 所示。

图 14.14　建立完成后的模型图

为了观察模型的输入，在此还需要加入若干辅助的模块。

(13) 从 Simulink 的 Sinks 模块库中复制 Scope 模块到 model14to01.mdl 模型中。若 Scope 被直接连接到电压输出端口进行测量，显示的将是伏特表示的电压。然而电工更习惯规范化的表达方式，即电压被表示成一个基准电压与此基准电压上下波动电压之和。

(14) 从 Simulink 的 Math Operations 模块库复制一个 Gain 模块到 model14to01.mdl 模型中，并设置增益为 $\dfrac{1}{424.4\times10^3\times\sqrt{2}}$。

(15) 从 Measurements 模块库中复制两个 Voltage Measurement 模块到 model14to01.mdl 模型中。

(16) 连接模型，得到最后完整的模型如图 14.15 所示。

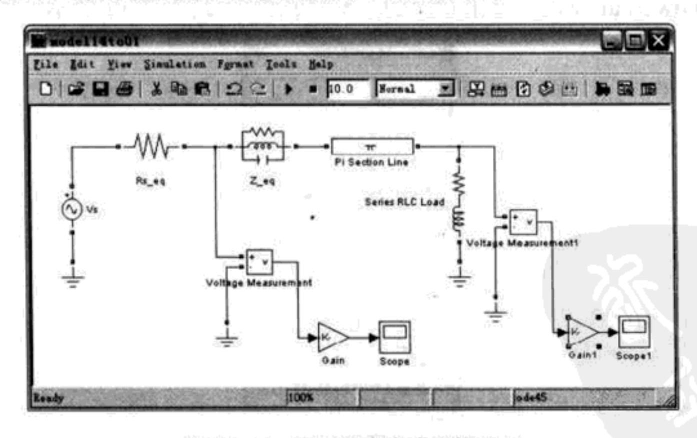


图 14.15　最后设置完成的模型图

(17) 进行模型的参数设置，由于此系统偏刚性，需要选择求解器。选择 Simulink 窗口的 Simulation|Configuration Parameters 命令。在弹出参数对话框中设置 Solver 选项中的 Solver 为 ode15s，Data Inport/Export 的 Limit Data Points to last 选项处于非选中状态，应用

设置。

(18) 进行模型仿真，得到结果如图 14.16 所示。

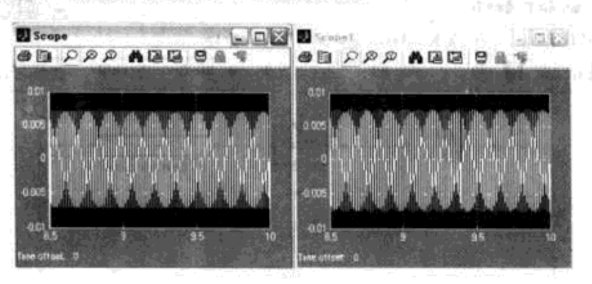


图 14.16　系统仿真结果

14.2.2　分析电路模型

对系统进行的分析涉及 4 个方面：

- 使用 Powergui 模块。
- 获取系统的稳定状态输出。
- 利用 power_analyze 命令分析电路系统。
- 在频域内分析电路系统。

下面来详细介绍稳定状态输出和频域内分析的操作及具体内容，其他方面会包含在这两个内容里面。

1. 稳定状态分析

(1) 为了便于对电路系统的稳定状态进行分析，powerlib 模块库中提供了一个 GUI(powergui)模块，将 powergui 模块复制到 model14to01.mdl 模型中。

(2) 双击 powergui 模块，弹出相应的参数对话框，如图 14.17 所示。

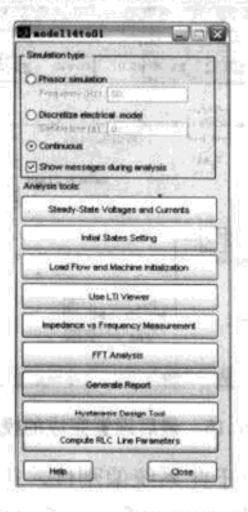


图 14.17　powergui 模块参数对话框

(3) 从 Powergui 对话框中的 Analysis Tools 组中单击 Steady-State Voltages and Currents 按钮，就会打开 Steady-State Tool 窗口，如图 14.18 所示，其中的两个测量结果由模型中的两个测量模块获取，以坐标的形式表示，并可从测量名称中知道相应的模块名称；

(4) 如果要显示稳定状态，可以选中 States 复选框。在此将所有的复选框全部选中，以便了解每一项的具体意义，如图 14.19 所示。

说明：窗口中显示的信息有赖于建模时对模块的操作顺序。状态的名称包含了模块名称和一个前缀，Il_表示感应电流，Uc_表示的是电容电压。

(5) 获取初始状态。

在如图 14.17 的对话框中单击 Initial State Settings 按钮，Steady-State Tool 窗口中就会显示 6 个状态变量，如图 14.20 所示。

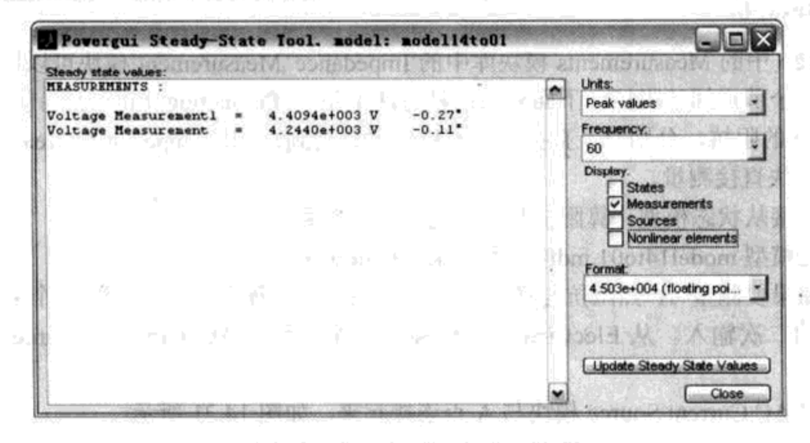


图 14.18 Steady-State Tool 窗口

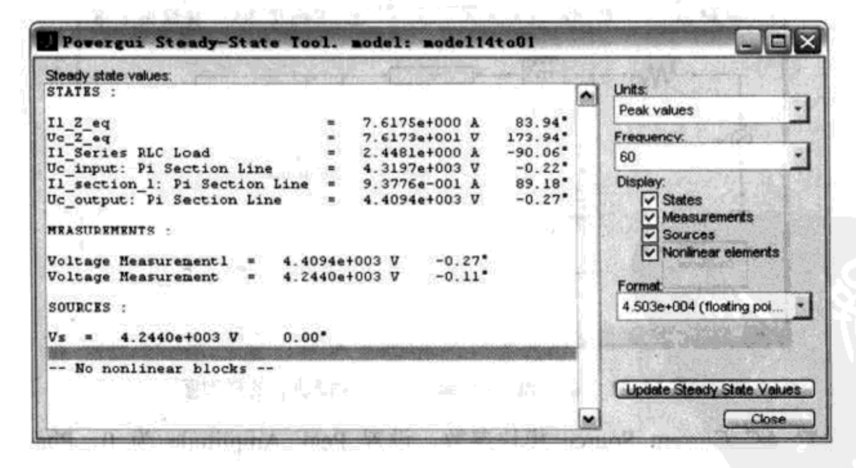


图 14.19 系统的各种信息显示

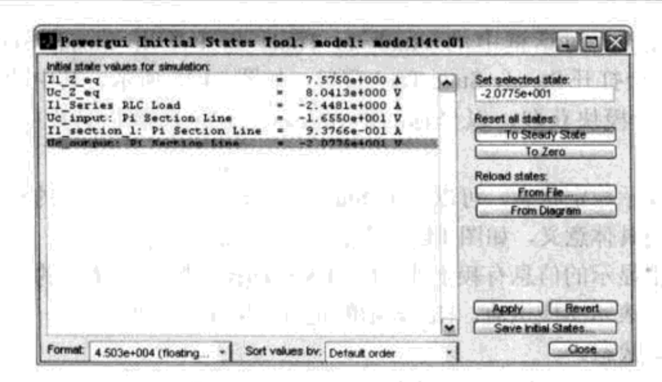


图 14.20 系统状态的初始值

2. 频域分析

Powerlib 中的 Measurements 模块库中的 Impedance Measurement 模块可以用来测量线圈中任何两个节点间的阻抗。下面介绍两种方法来测量 Pi Section Line 右端节点(设为 A 点)与地之间的阻抗，分别是：直接从状态模型计算得到；利用 Impedance Measurement 和 Powergui 模块直接测量。

1) 直接从状态模型计算阻抗与系统的频率的关系

(1) 将模型 model14to01.mdl 另存为 model14to02.mdl。

(2) 如果要测量 A 点阻抗与频率之间的关系，还必须在模型中添加一个电流源作为状态模型的二次输入。从 Electrical Sources 模块库中复制 AC Current Source 模块到模型中。

(3) 将 AC Current Source 模块与 A 点连接起来，如图 14.21 所示。

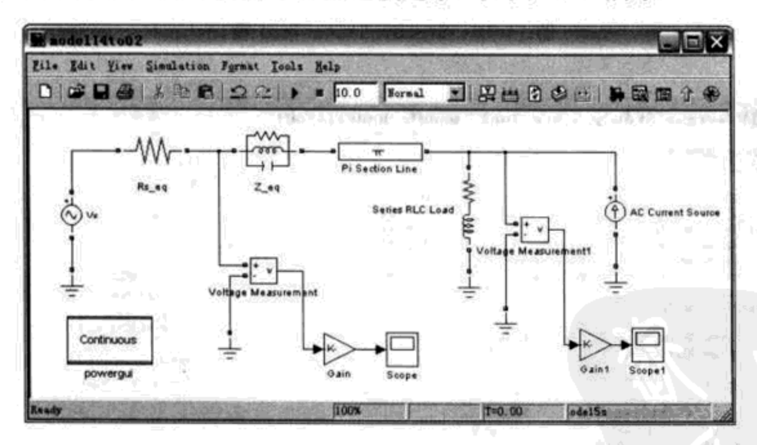


图 14.21 添加 AC Current Source 模块后的模型图

(4) 设置 AC Current Source 模块参数。设置 Peak Amplitude 为 0，Phase 为 0，Frequency 为 60。

(5) 利用 power_analyze 命令计算 model14to02.mdl 模型的状态，在 MATLAB 命令窗口中输入：

```
>> [A,B,C,D,x0,states,inputs,outputs]=power_analyze('model14to02');
```

然后在 MATLAB 命令窗口中输入想要得到的状态变量，可以得到相应的结果如下：

```
>> states,inputs,outputs
states =
Il_Z_eq
Uc_Z_eq
Il_Series RLC Load
Uc_input: Pi Section Line
Il_section_1: Pi Section Line
Uc_output: Pi Section Line
inputs =
U_Vs
I_AC Current Source

outputs =
U_Voltage Measurement1
U_Voltage Measurement
```

说明：　这些变量和状态可以直接从 powergui 中得到。

(6)　得到了系统的状态模型之后，就可以进行频域分析了。例如电路的模态就可以通过对矩阵 A 进行特征分析得到，在 MATLAB 命令窗口中输入。

```
>> eig(A)
ans =
  1.0e+005 *
  -2.4972
  -0.0001 + 0.0144i
  -0.0001 - 0.0144i
  -0.0002 + 0.0056i
  -0.0002 - 0.0056i
  -0.0000
```

(7)　在 Laplace 域内，A 点阻抗 Z 可以由 A 点的电流 I 与 A 点测量的电压 U 决定，关系如下：$Z=\dfrac{U}{I}$

0 到 1500 Hz 的节点 A 处的阻抗可以通过下列程序计算，得到结果如图 14.22 所示。

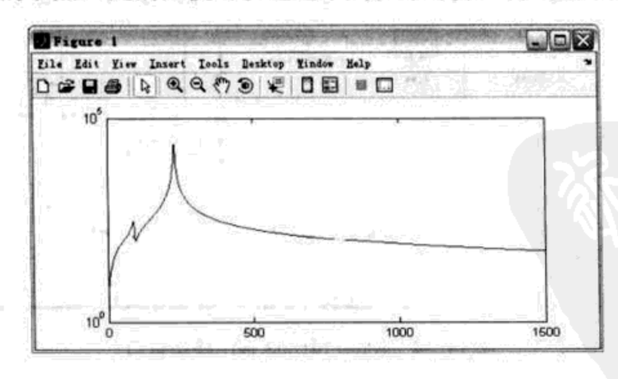


图 14.22　Number of sections 为 1 情况分析结果

```
[A,B,C,D,x0,states,inputs,outputs]=power_analyze('model14to02');
freq=0:1500;
w=2*pi*freq;
```

```
[mag1,phase1]=bode(A,B,C,D,2,w);
semilogy(freq,mag1(:,2));
```

(8) 双击 PI Section Line 模块，将对话框中的参数 Number of sections 由原来的 1 改为 10。为了得到，Number of sections 为 1 和 10 两种情况的分析结果，可输入如下命令，得到的结果如图 14.23 所示。

```
[A,B,C,D]=power_analyze('model14to02');
[mag10,phase10]=bode(A,B,C,D,2,w);
semilogy(freq,mag1(:,2),freq,mag10(:,2));
grid on
```

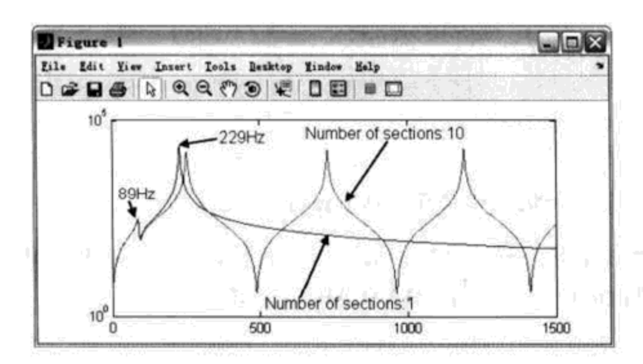


图 14.23 Number of sections 为 1 和 10 两种情况的分析结果

2) 利用 Impedance Measurement 和 Powergui 模块直接测量

在 SimPowerSystems 中，电路阻抗可以自动测量，而不需要计算。

(1) 打开 model14to01.mdl，并另存为 model14to03.mdl。

(2) 从 powerlib 库中 Measurements 模块库中复制 Impedance Measurement 模块到模型 model14to03.mdl，连接模块如图 14.24 所示。

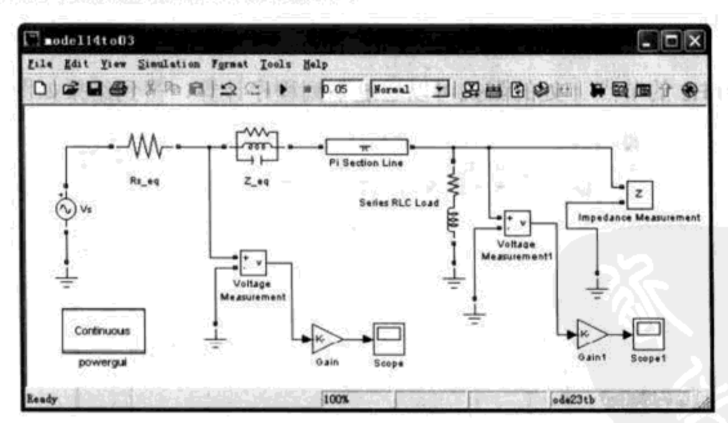


图 14.24 model14to03.mdl 模型窗口

(3) 双击 powergui 模块，在 Tools menu 中单击 Impedance vs Frequency Measurement 选项，弹出如图 14.25 所示数据显示窗口。

(4) 选择 model14to03.mdl 模型窗口的 Simulation|Configuration parameters 命令。在

Solver 控制面板中，设置求解器为 ode3tb，设置 Relative tolerance 为 1e-4，设置 Stop time 为 0.05。

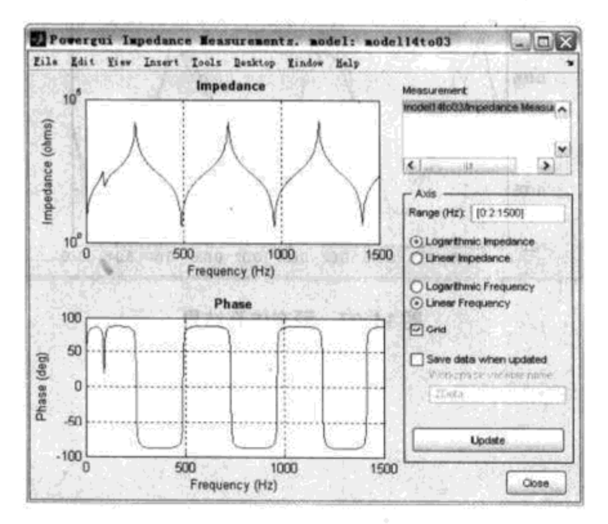


图 14.25　结果分析窗口

(5) 进行仿真，打开 Scope 模块，查看运行结果。

(6) 然后双击 Powergui 模块，在弹出对话框中单击 Initial State Settings 按钮，然后在弹出对话框中单击 To zero 按钮使得所有状态为 0，最后单击 Apply 按钮。

(7) 设置 Scope 模块，单击 Scope 窗口的 Parameters 按钮，在弹出窗口中选择 Data History 标签，选中 Save data to workspace 复选框，如图 14.26 所示。同样设置 Scope1 模块。

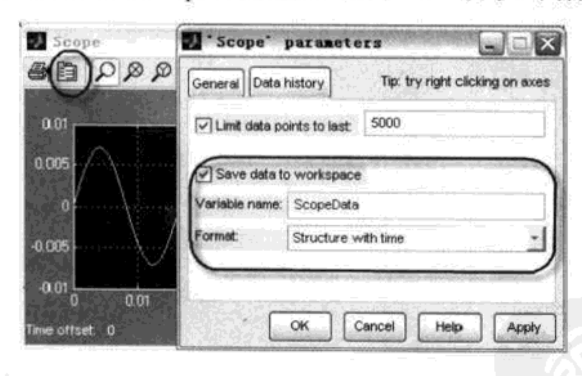


图 14.26　Scope 设置

(8) 运行仿真。

(9) 在 MATLAB 命令窗口输入：

```
>> plot(ScopeData.time,ScopeData.signals.values,ScopeData1.
time,ScopeData1.signals.values)
```

得到结果如图 14.27 所示，这是系统的瞬态仿真结果。

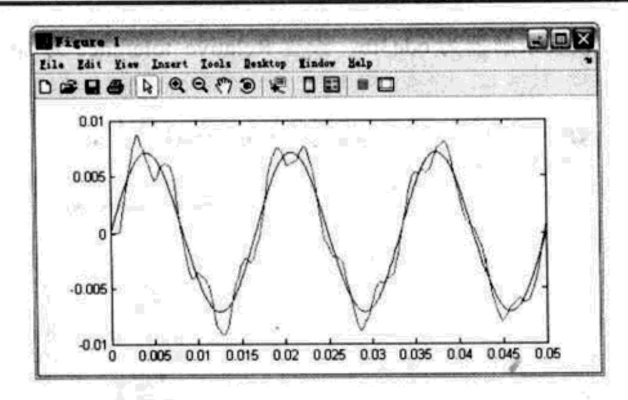


图 14.27 瞬态仿真结果

第 15 章　Simulink 控制设计工具箱

在利用 Simulink 控制设计工具箱之前应该了解这个工具箱的功能和主要的应用领域，这有助于激发对此工具箱的兴趣。然后介绍利用工具箱对模型进行线性化的基本步骤。最后通过一个具体的实例，来讲解整个工具箱的功能，实现的基本步骤等。

本章主要内容包括：

- Simulink 控制系统设计
- 线性化模型
- 磁力球模型线性化实例

15.1　Simulink 控制系统设计

Simulink Control Design 为 Simulink 模型中的控制系统和受控系统提供了线性化工具。线性化 Simulink 模型通常可以简化系统分析和补偿设计。这在很多方面都有应用，具体如下：

- 航空宇航：飞行控制，制导与导航。
- 汽车自动化：自动驾驶，启动和运行控制。
- 仪器制造：发动机，磁盘驱动和伺服器。

Simulink Control Design 主要是利用 Control System Toolbox 工具箱的一些功能，例如 LTI Viewer、SISO Design Tool 和 Simulink 线性化引擎。Simulink Control Design 可以用来：

- 无需在模型中插入模块就可获得输入输出点。
- 无需删除反馈控制环就可以进行开环分析。
- 从仿真中获取工作点。
- 从隐含定义的状态和输出中计算工作点。
- 检查和调试线性化结果。
- 将结果输出到 MATLAB 工作空间。
- 管理线性化工程。
- 使用新的线性化和调整函数。

Simulink Control Design GUI 为控制系统线性化和设计提供了图形界面交互环境。通过 GUI 可以非常容易地检查和分析工作点以及线性化结果。此外，还可以保存和恢复设置以及将结果输出到 MATLAB 工作空间。

同样可以通过 MATLAB 命令得到线性化模型。利用函数，可以自动处理线性化过程，执行批处理。例如，同一个系统在不同参数下的线性化。

15.2 线性化模型

利用 Simulink Control Design GUI 来线性化 Simulink 模型。利用这个工具用户可以插入、检查和改变输入输出点而不用另外添加模块到模型当中。还可以不用手动断开信号线直接进行开环系统分析，确定和计算线性化的工作点。此外，还可以保存和恢复设置，以及将结果输出到 MATLAB 工作空间，并利用 LTI Viewer 查看结果。

利用 Simulink Control Design GUI 来线性化 Simulink 模型的主要操作步骤如下：

(1) 创建或打开一个模型。

(2) 在 Control and Estimation Tools Manager 中打开一个线性化工程。

(3) 设置模型。

(4) 确定工作点。

(5) 线性化模型或模块。

(6) 分析结果。

(7) 导入并保存结果。

后面就介绍一个带有非线性系统的例子，在其随后的小节中，就是用这个例子讨论每一步如何操作。

15.3 磁力球模型线性化实例

15.3.1 磁力球模型示意图

电路如图 15.1 所示，由一个电源、电阻和一个线圈电感组成。一个铁球在电感的正下方，同时受到重力和磁力的作用。

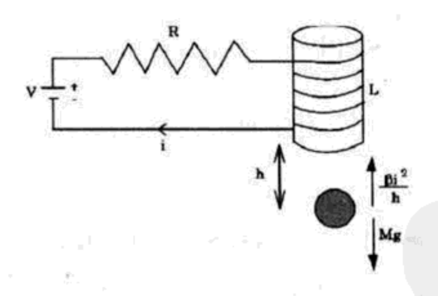


图 15.1 电路系统示意图

15.3.2 磁力球模型方程

此方程描述球受到非平衡力作用时的高度 h 的变化，系统的动力学方程：$m\frac{d^2h}{dt^2}=mg-\frac{\beta i^2}{h}$，其中 m 为球质量，h 是球高度，g 是重力加速度，i 为电流，β 是与磁力相关的常数。

第 15 章　Simulink 控制设计工具箱

在利用 Simulink 控制设计工具箱之前应该了解这个工具箱的功能和主要的应用领域，这有助于激发对此工具箱的兴趣。然后介绍利用工具箱对模型进行线性化的基本步骤。最后通过一个具体的实例，来讲解整个工具箱的功能，实现的基本步骤等。

本章主要内容包括：

- Simulink 控制系统设计
- 线性化模型
- 磁力球模型线性化实例

15.1　Simulink 控制系统设计

Simulink Control Design 为 Simulink 模型中的控制系统和受控系统提供了线性化工具。线性化 Simulink 模型通常可以简化系统分析和补偿设计。这在很多方面都有应用，具体如下：

- 航空宇航：飞行控制，制导与导航。
- 汽车自动化：自动驾驶，启动和运行控制。
- 仪器制造：发动机，磁盘驱动和伺服器。

Simulink Control Design 主要是利用 Control System Toolbox 工具箱的一些功能，例如 LTI Viewer、SISO Design Tool 和 Simulink 线性化引擎。Simulink Control Design 可以用来：

- 无需在模型中插入模块就可获得输入输出点。
- 无需删除反馈控制环就可以进行开环分析。
- 从仿真中获取工作点。
- 从隐含定义的状态和输出中计算工作点。
- 检查和调试线性化结果。
- 将结果输出到 MATLAB 工作空间。
- 管理线性化工程。
- 使用新的线性化和调整函数。

Simulink Control Design GUI 为控制系统线性化和设计提供了图形界面交互环境。通过 GUI 可以非常容易地检查和分析工作点以及线性化结果。此外，还可以保存和恢复设置以及将结果输出到 MATLAB 工作空间。

同样可以通过 MATLAB 命令得到线性化模型。利用函数，可以自动处理线性化过程，执行批处理。例如，同一个系统在不同参数下的线性化。

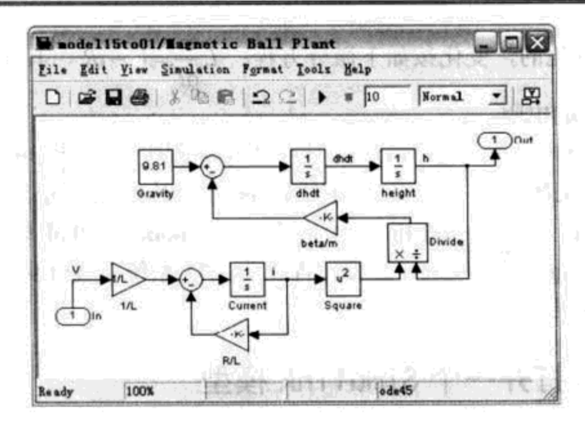


图 15.3 Magnetic Ball Plant 子系统

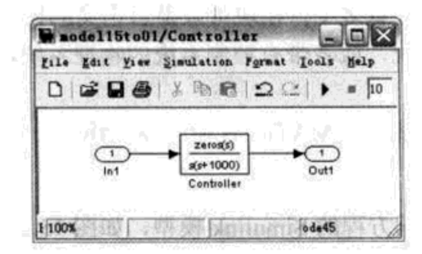


图 15.4 控制子系统

15.3.4 开始线性化工程

Control and Estimation Tools Manager 提供了一个图形化的环境来执行和管理、控制和估计线性化。在这个环境中可以创建一个工程，工程中可以包含多个任务。

在 Simulink 模型空间中选择 Tools | Control Design | Linear Analysis 命令，或者在 MATLAB Start 菜单中选择 Start | Simulink | Simulink Control Design | Linear Analysis Tool 命令。打开 Control and Estimation Tools Manager 窗口，如图 15.5 所示。

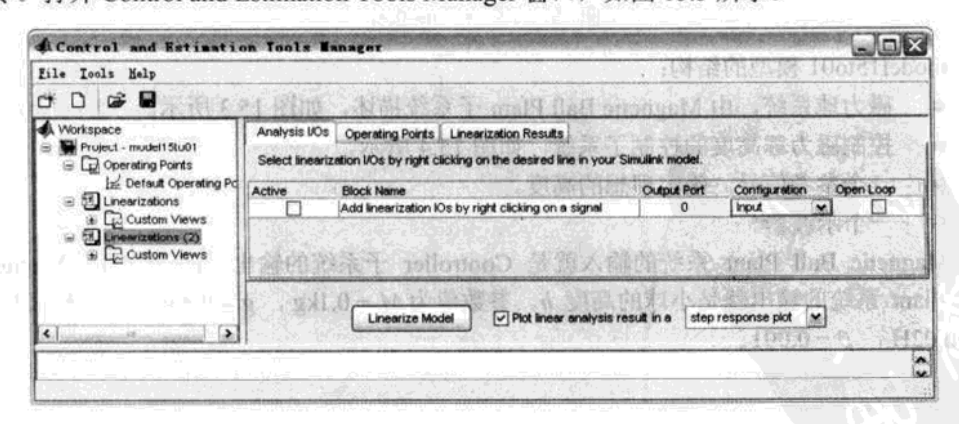


图 15.5 Control and Estimation Tools Manager 窗口

Control and Estimation Tools Manager 窗口可以创建并管理工程。左边的控制面板是工程目录树，包含了所有目前的工程。目前只有一个工程 model15to01。选择一个节点，在右边的窗口查看内容。

15.3.5　配置一个线性化模型

在模型线性化之前必须配置模型，包括选择输入输出线性化点。然后，如果需要进行开环系统分析，还必须插入开环点。单击 Control and Estimation Tools Manager 中的 Linearization 节点，可利用 Analysis I/Os 控制面板来检查被选择的线性化点。

线性化输入点为被线性化系统定义一个输入点，而线性化输出点为被线性化系统定义一个输出点。此外在线性化模型计算扰动时，可以在扰动输入的地方添加一个输入点，在扰动测量的地方添加一个输出点。

输入点与输出点之间的区域即为需要线性化的部分，除非有一个反馈环将输出信号返回到输入信号点。如果系统中存在这么一个闭环，就不得不通过添加一个开环点来去除闭环效应。

说明： 不应该将线性化输入输出点与 Simulink 的 Inport 和 Outport 模块搞混淆。线性化输入输出点定义了线性分析系统的输入与输出，而 Inport 和 Outport 模块则定义了整个系统的输入与输出。

1. 选择线性化点

可以用 Simulink Control Design 来线性化整个模型、模型中任何一个模块或者子系统。在需要线性化部分的前面和后面插入一个线性化点。

在整个模型中，只有 Magnetic Ball Plant 系统是非线性的，线性化这个部分的操作步骤如下：

(1) 在 Control and Estimation Tools Manager 中选择线性化节点。

(2) 在 Magnetic Ball Plant 子系统和 Controller 子系统之间的信号线上右击，在弹出的菜单中选择 Linearation | Input Point 命令，就会出现一个相应的标志，如图 15.6 所示。

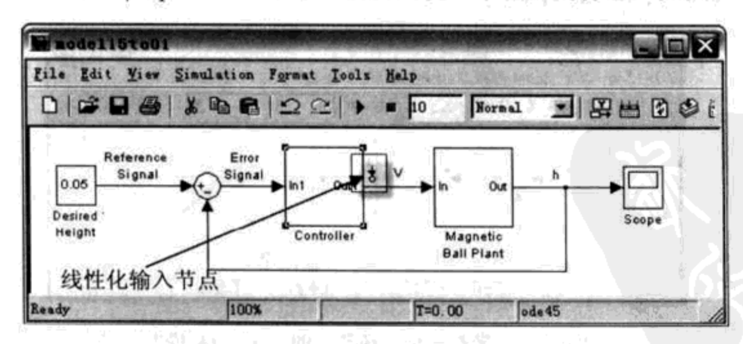


图 15.6　添加输入节点后的 Simulink 模型

(3) 在 Magnetic Ball Plant 子系统后的信号线上右击，在弹出的菜单中选择 Linearation | Output Point 命令，就会出现一个相应的标志，如图 15.7 所示。

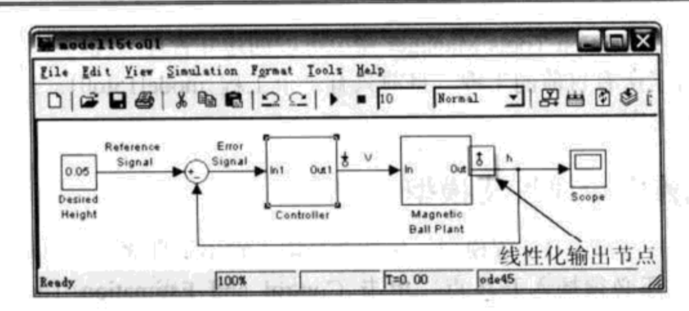


图 15.7 添加输出节点的 Simulink 模型

说明： 可以通过 Linearization Points 菜单中的选项创建一个更加复杂的线性化模型，可以获得传递函数等。

- Input-Output Point：输入节点之后紧跟一个输出节点，这主要用来测量输出扰动的敏感度。
- Output-Input Point：一个输出节点后面紧跟一个输入节点，这对鲁棒控制有用。
- Open Loop 和 Output Constraint 将在后面的讨论。

2. 删除线性化节点

如果要删除某信号线上的线性化节点，可以重复上面的过程。例如删除输入节点，操作为：在 Magnetic Ball Plant 子系统和 Controller 子系统之间的信号线上右击，在弹出的菜单中选择 Linearation | Input Point 命令。

3. 进行开环分析

由于模型中存在反馈环，输入输出节点可能不能完全定义模型的线性化部分。为了能够消除反馈信号的影响，必须插入一个开环点。

为了能够只对 Magnetic Ball Plant 子系统线性化，右击输出节点的信号线，单击 Linearation | Open Loop 选项。就会在输出节点的后面会出现一个 x 标记，表示已经是一个开环系统了，如图 15.8 所示。

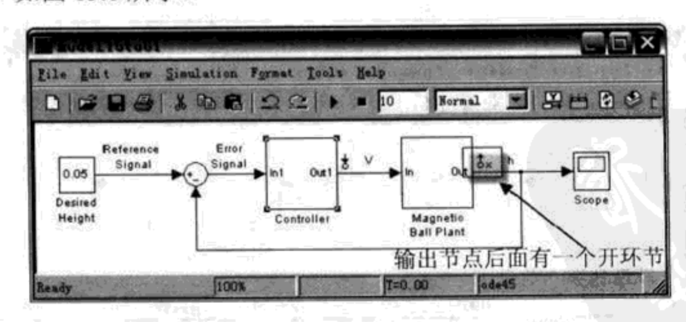


图 15.8 带有一个开环节点的 Simulink 模型

4. 查看输入输出分析

为了查看线性化点，可以在 Control and Estimation Tools Manager 窗口中选择 Linearization 节点中的 Analysis I/Os 控制面板。利用这个面板来查看和修改线性化点。如

图 15.9 所示。

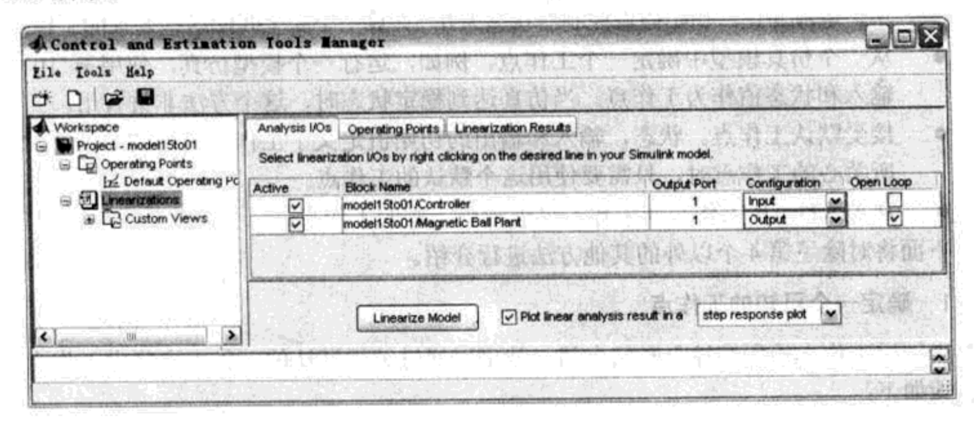


图 15.9 Control and Estimation Tools Manager 窗口

说明： 当选择 Numerical Perturbation 作为线性化算法时，是不能对 Analysis I/Os 进行修改，或是执行开环分析的。可以在 Control and Estimation Tools Manager 窗口中选择 Tools | Optionswgwy 命令，打开 Options 对话框，如图 15.10 所示。

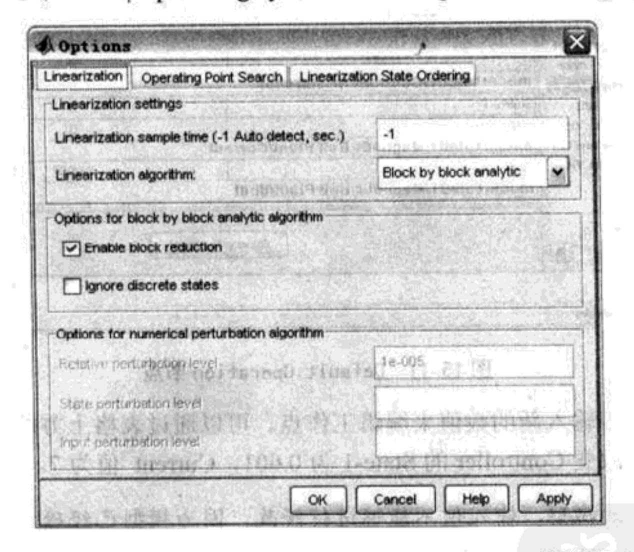


图 15.10 Control and Estimation Tools Manager 选项窗口

15.3.6 确定工作点

在线性化之前，必须选择一个线性系统工作点，通常是指稳态值。共有 5 种方法可以找到系统的工作点。

- 完全制定工作点的输入和状态。例如，知道磁力球的高度为 0.05，高度的变化率为 0，电流应该为 7.0036，还知道控制器中的状态值，这些值在工作点处被完全确定。
- 确定目标值以及输入、输出和状态子集的约束。Simulink Control Design 利用数

值方法确定全部的工作点。例如，知道磁力球的高度为 0.05，高度的变化律很小，电流为正，初始假设控制器状态为 0，计算一个完全匹配这些要求的工作点。
- 从一个仿真模型中确定一个工作点。例如，运行一个模型仿真，使用第 10 秒的输入和状态值作为工作点。当仿真达到稳定状态时，这个方法非常有用。
- 接受默认工作点。状态，输入和输出的初始值定义了工作点。当初始值非常接近所关心的工作点时，只需要使用这个默认的工作点。
- 从 MATLAB 工作空间中导入另外一个工程的工作点。

下面将对除了第 4 个以外的其他方法进行介绍。

1. 确定一个已知的工作点

当知道工作点的所有输入和状态值，就可以通过编辑默认的工作点来确定工作点，操作步骤如下：

(1) 在 Control and Estimation Tools Manager 窗口中选择 Operating Points 下的 Default Operation 节点，如图 15.11 所示。

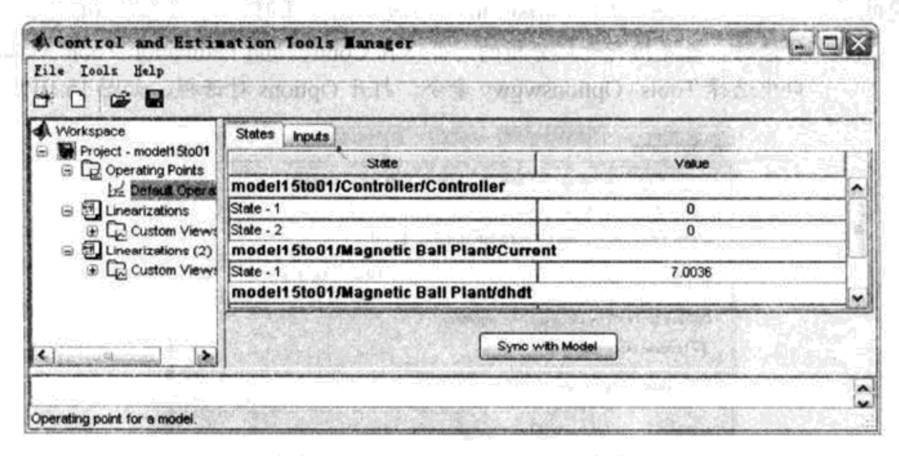


图 15.11 Default Operation 节点

(2) 通过在表中输入新的数值来编辑工作点。可以通过表格上方的标签在状态和输入之间切换。例如，改变 Controller 的 State-1 为 0.001，Current 值为 7。

注意： 对于此模型，输入值不能够进行修改，因为模型已经确定了所有的输入。

当需要添加状态，输入或者输出，或者将它们从模型中删除，单击 Sync with Model 按钮来更新工作表设置。

2. 通过给定要求确定工作点

1) 计算工作点

当用户只知道部分的或是隐含的信息时，通过 Simulink Control Design 工具利用一些确定的要求计算出工作点。

(1) 建立一个新的操作点。在 Control and Estimation Tools Manager 窗口中选择 Operating Points 节点，然后选择 Create Operating Points 控制面板。

(2) 在 Compute new operating points using 下拉列表框中选择 operating specifications 选项，则 Control and Estimation Tools Manager 窗口如图 15.12 所示。

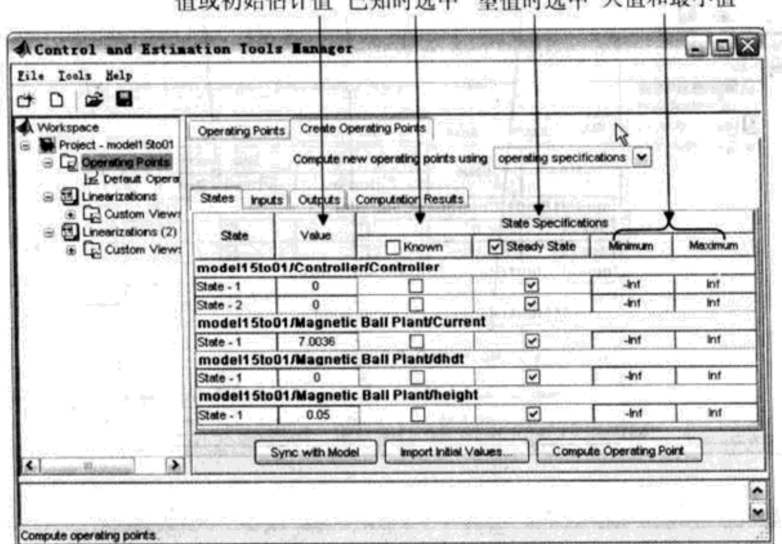


图 15.12　Control and Estimation Tools Manager 窗口

(3) 在表格中输入工作点的要求，例如任何知道的值或约束。可以在状态、输入和输出之间转换。

图 15.13 显示了关于磁力球模型的一系列要求。在这个实例中，输入和输出都是确定的。

说明： 当添加状态、输入和输出的时候，可以单击 Sync with Model 按钮，这个按钮还可以用来删除状态、输入和输出。

(4) 计算工作点，单击 Compute Operating Point 按钮，Simulink Control Design 就会寻找一个最匹配给定要求的工作点，并添加一个新的工作点，标签为 Operating Point。

注意： 某些近似确定的参数可能不是精确地满足要求，具体如图 15.14 所示。

2) 导入工作点

如需要从另外一个工作点、Simulink 状态或向量值中导入一个工作点，单击图 15.13 中的 Import Initial Values…按钮，会打开 Operating Point Import 参数对话框，如图 15.15 所示：

在这个对话框中，可以选择导入数据的类型，从一个工程中、工作空间中还是文件中导入数据，然后从可供选择的工作点列表中选择工作点。单击 Import 按钮，将初始数据导入工作点要求的表格中。

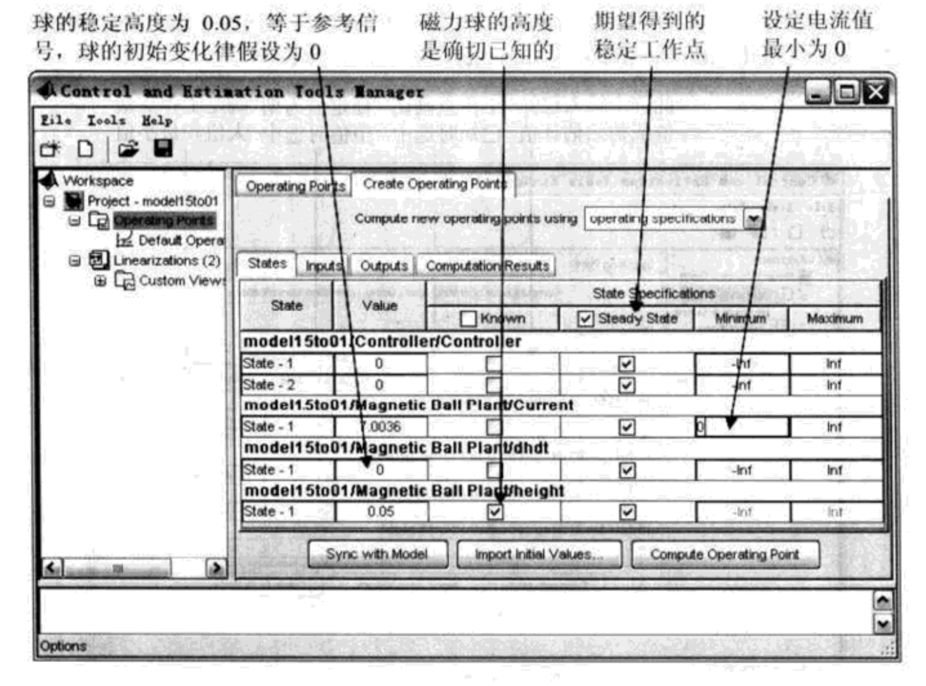


图 15.13 磁力球模型的一系列设置

dx 很小表示找不到稳定值

Control and Estimation Tools Manager

State Name	Desired Value	Actual Value	Desired dx	Actual dx
model15to01/Controller/Controller				
State - 1	[-Inf , Inf]	0	0	0
State - 2	[-Inf , Inf]	-2.5561e-006	0	0
model15to01/Magnetic Ball Plant/Current				
State - 1	[-Inf , Inf]	7.0036	0	3.4447e-011
model15to01/Magnetic Ball Plant/dhdt				
State - 1	[-Inf , Inf]	0	0	-1.7453e-010
model15to01/Magnetic Ball Plant/height				
State - 1	0.05	0.05	0	0

- An operating point called "Operating Point" has been added to the node Operating Points.

Operating points for a model.

图 15.14 工作点计算结果

Operating Point Import

Import from:

Select from project　Project - model15to01

Workspace

MAT-file:　Browse

Available Data	Type	Size
Default Operating Point	Operating Point	5 - States
Operating Point	Operating Point	5 - States

Import　Close　Help

图 15.15　Operating Point Import 参数对话框

3)　约束输出

工作点的确定通常包括模型中某些特定信号值的约束。通过给定要求确定工作点，实时对输出进行约束，在需要的信号线上右击，在弹出的菜单中选择 Output Constraint 命令，就可以在相应的地方添加一个输出约束标记 T，如图 15.16 所示。Create Operating Points 面板中的 Outputs 面板可以设定输出的最大值和最小值，如图 15.17 所示。

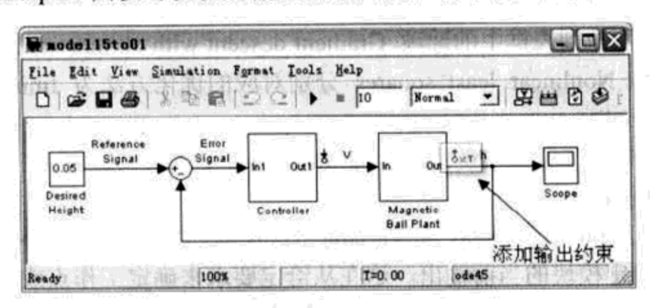


图 15.16　添加输出约束的 Simulink 模型

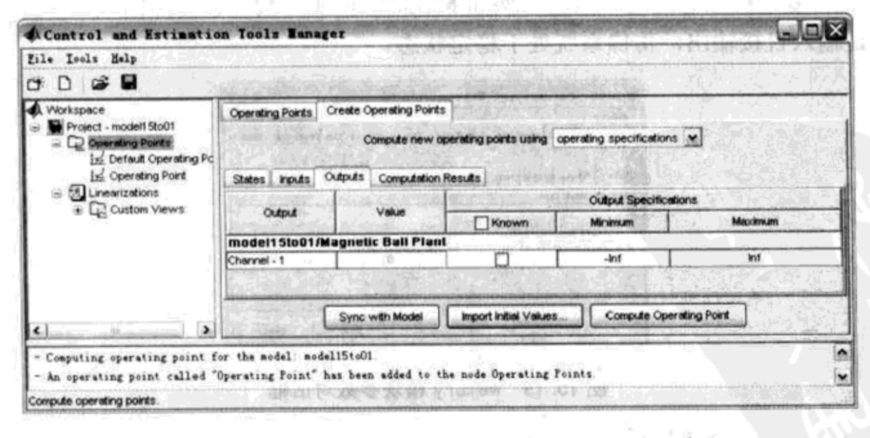


图 15.17　Control and Estimation Tools Manager 窗口

4) 改变优化设置

使用最佳化方法确定工作点时，用户可以改变默认设置。在 Control and Estimation Tools Manager 窗口中选择 Tools | Options 命令，在弹出的参数对话框中选择 Operating Point Search 选项卡，如图 15.18。

图 15.18 Linearization Task Options 对话框

其中 Method 下拉列表框中的选项 Gradient descent with elimination, Gradient descent, Simplex search 和 Nonlinear least squares 分别对应的优化方法为 fmincon, graddescen, fminsearch 和 lsqnonlin。

Options 对话框中的选项对应于函数 optimset 的各个参数设置。

5) 使用非直通模块

例如 Memory, Transport Delay 和 Variable Transport Delay 模块都不能够直通。模块的当前输入并不能确定模块的当前输出。这在从给定要求来确定工作点或者创建线性化模型时会产生问题。可以选中这些模块参数对话框中的 Direct feed through of input during linearization 复选框来避免这个问题。例如 Memory 模块参数对话框如图 15.19 所示。这个选项就是输入直接输出，仿佛系统处于稳定状态。

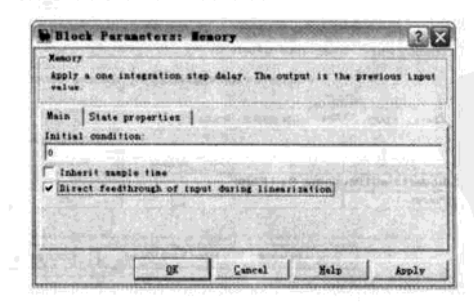


图 15.19 Memory 模块参数对话框

3. 从仿真中获取工作点

利用 Simulink Control Design 在指定的时刻从仿真中获取工作点。例如仿真到达稳定

状态时的解，操作步骤如下：

(1) 创建一个新的工作点，在 Control and Estimation Tools Manager 窗口中选择 Operating Points 节点，然后选择 Create Operating Points 选项卡。

(2) 在 Create Operating Points 选项卡中，设置 Compute new operating points using 下拉列表框为 simulation snapshots，如图 15.20 所示。

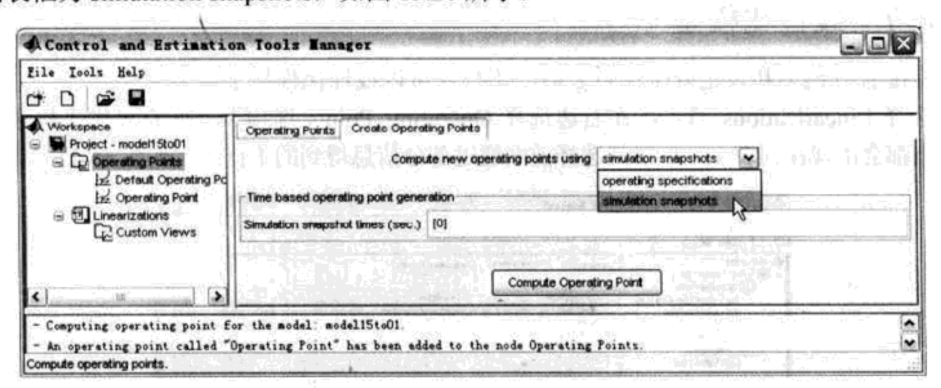


图 15.20　设置 Create Operating Points 选项卡

(3) 在 Simulation snapshot times 文本框中输入一个时间向量，例如 t=1 和 t=10 时刻的工作点，可输入[1,10]。

(4) 单击图 15.20 中的 Compute Operating Point 按钮，Simulink Control Design 将对模型进行仿真，并获取工作点。在工程数菜单中的 Operating Points 节点下面添加两个新节点，标签分别为 Operating Point at t=1 和 Operating Point at t=10，选择其中一个节点进行查看，如图 15.21 所示。

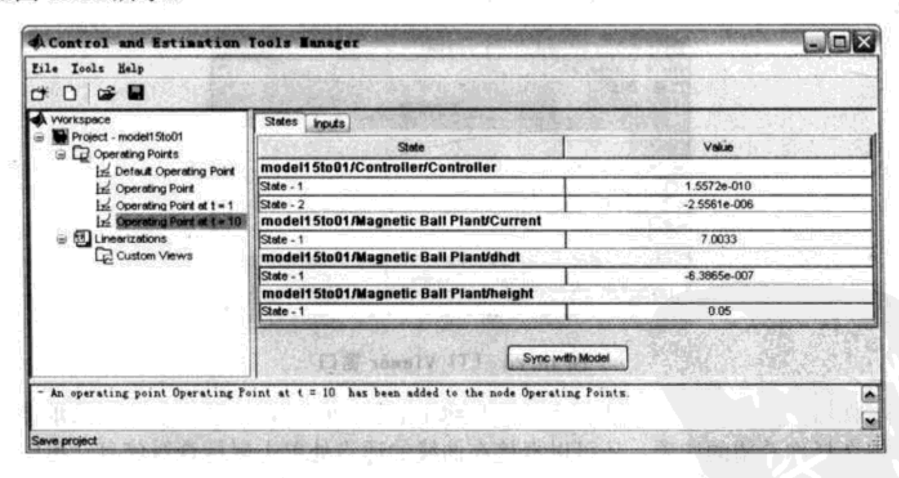


图 15.21　Simulation Snapshot Operating Point 结果

4. 导入工作点

利用 Simulink Control Design 从 MATLAB 工作空间或者 MAT 文件中导入工作点，操作步骤如下：

(1) 为了能够导入一个新的工作点，可选择 Operating Points 节点，然后选择右边的 Operating Points 选项卡，单击下方的 Import 按钮，弹出 Operating Point Import 参数对话框。

(2) 在 Operating Point Import 对话框中，选择 Workspace 或者 MAT-file 选项，就可以确定导入的形式，然后从下面的列表中选择一个工作点，最后单击 Import 按钮导入。

15.3.7 线性化模型

在配置好模型和设定好工作点之后，用户就可以线性化模型了。

选择 Linearizations 节点，在右边选择 Operating Points 选项卡。所有可供工程选择的工作点都会出现在列表中。这里就选择前面通过部分信息得到的工作点，如图 15.22 所示。



图 15.22 选择线性化工作点

单击图 15.22 底部的 Linearize Model 按钮来计算线性化模型。默认情况下，新的线性化系统受到一个阶越激励，并在 LTI Viewer 窗口显示结果，如图 15.23 所示。操作步骤如下：

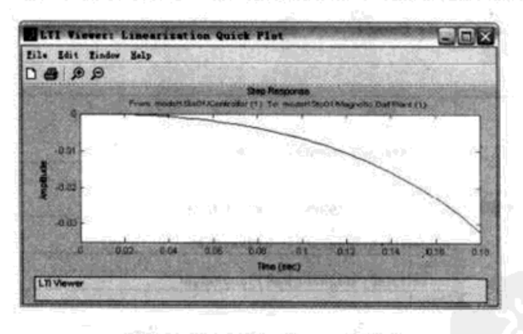


图 15.23 LTI Viewer 窗口

(1) 使用 LTI Viewer。为了在图中添加峰值和稳态时间，在图形窗口任何地方右击，从菜单中选择你希望的选项。还可以直接在曲线的适当处单击鼠标查看信息。可以同时一次显示多达 6 个图形，可以在 Edit | Plot Configurations 中设置。通过选择 File | Export 把线性化模型导入到工作空间。

(2) 修改线性化设置。可以通过选择 Control and Estimation Tools Manager 窗口中的 Tools | Options 命令，在弹出的对话框中选择 Linearization 选项卡，如图 15.24 所示。

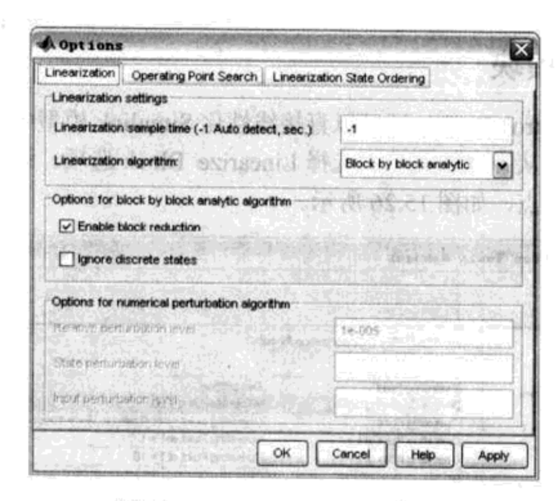


图 15.24 Linearization 选项卡

(3) 修改状态的顺序。修改线性化模型中状态向量中状态的顺序，这对比较不同模型的状态非常有用。可以通过选择 Control and Estimation Tools Manager 窗口中的 Tools | Options 命令，在弹出的对话框中选择 Linearization State Ordering 选项卡，如图 15.25 所示。

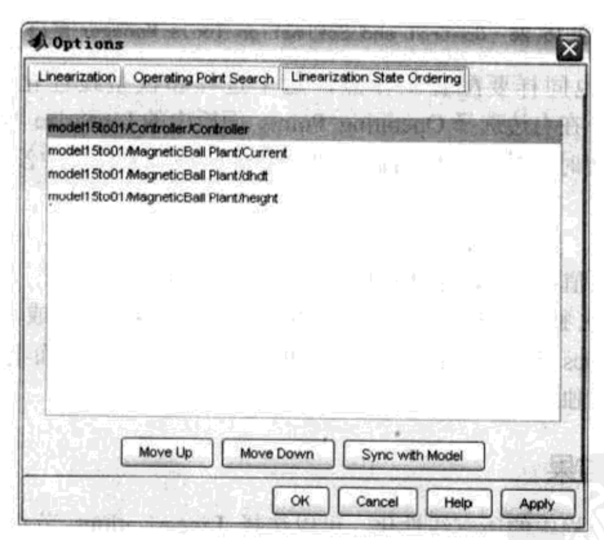


图 15.25 Linearization State Ordering 选项卡

使用 Move Up 和 Move Down 按钮移动状态到一个新的位置，如果需要增加状态，可以单击 Sync with Model 按钮。

(4) 线性化离散和多速率模型。对于含有离散状态或者多速率模型，线性化方法是一样的。然而这些线性化样本时间可以在 Linearization 选项卡中进行调整。默认情况下，这个参数为-1，这个使得 Simulink Control Design 以最低的样本速率进行线性化。可以在对话框中输入一个不同的样本时间来进行线性化。0 表示对一个连续模型进行线性化。

15.3.8 线性化模块

利用 Simulink Control Design 还可以直接线性化 Simulink 模型中的一个模块。可以右击要线性化的模块上，从弹出的菜单中选择 Linearize Block 选项。这就会在工程树目录中增加一个模块线性化节点，如图 15.26 所示。

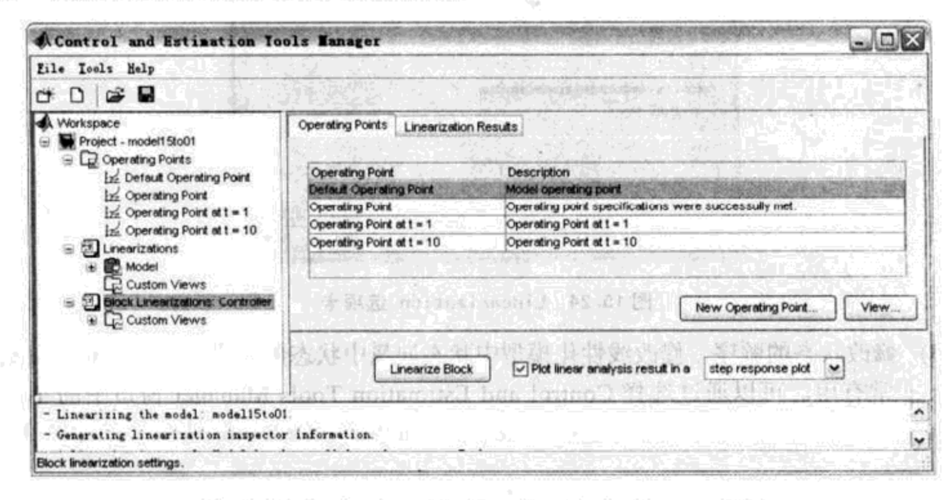


图 15.26 Control and Estimation Tools Manager 窗口

执行线性化，也同样要配置工作点，这个过程和模型线性化一样，单击 Block Linearizations 节点，在右边选择 Operating Points 面板中的 Linearize Block 按钮。不需要选择输入输出点，此时为默认模块的输入和输出端口。如果模型中没有线性化输入和输出，也就将被忽略。

注意事项：

- 不能使用数值扰动方法来线性化一个独立的模块。
- 为了线性化独立的模块，必须至少含有一个输入端口或者输出端口。由于 SimMechanics 和 SimPowerSystems 模块只是连接端口，而不是输入输出端口，所以不能单独地被线性化。

15.3.9 分析结果

为了显示当前工程中的所有线性化，可以选择 Linearizations 节点，然后在右边的窗口中选择 Linearization Results 选项卡。如果要删除 Linearization Results 列表中的某个结果，先选中它，然后单击选项卡中的 Delete 按钮。

1. 确认线性化结果

为了显示线性化结果，可以单击 Model 节点，如果要删除这个节点，则可以右击 Model，从弹出菜单中选择 Delete 命令。如图 15.27 所示。操作步骤如下：

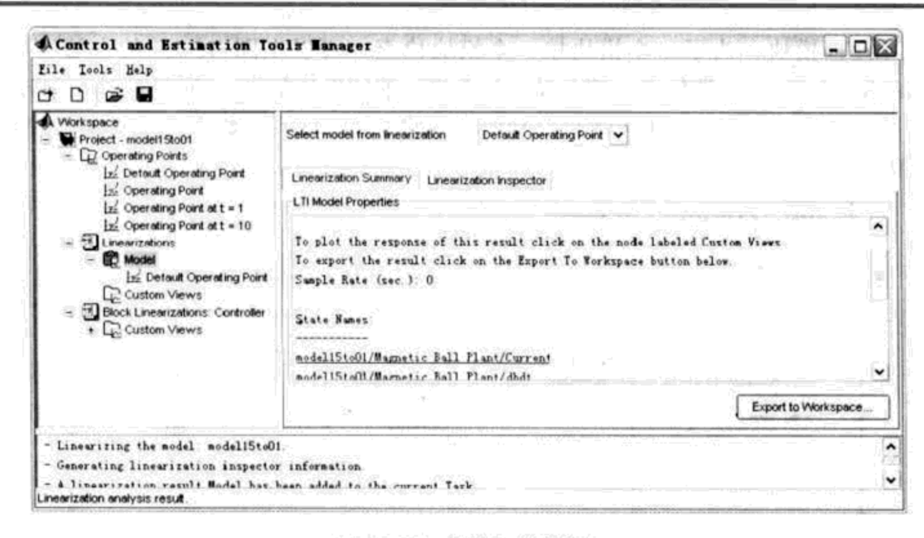


图 15.27 线性化模型简介

(1) 亮显线性化模块。为确定是否是线性化需要部分，可亮显线性化模块来实现。在 Control and Estimation Tools Manager 窗口中右击 Model 节点处，在弹出的菜单中选择 Highlight Blocks in Linearization 命令，会亮显线性化模块以及其的中子系统模块，如图 15.28 和 15.29 所示。

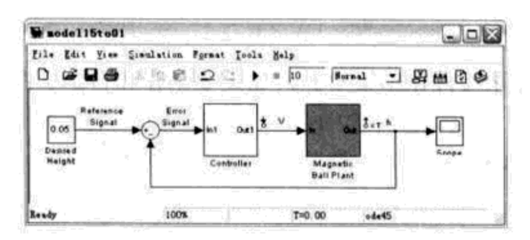


图 15.28 模型亮显

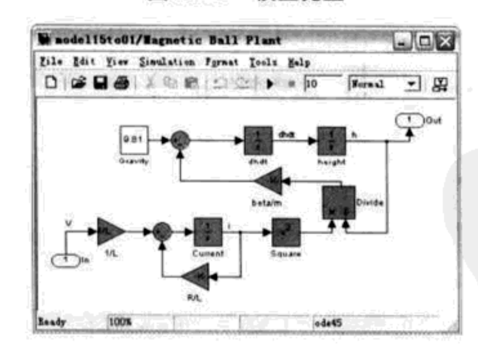


图 15.29 子系统亮显

注意：当使用数值扰动线性化方法，就不能够使用 Highlight Blocks in Linearization 或者 Remove Highlighting。

(2) 检查模块。为了一个模块接一个模块检查，可以在图 15.25 的右边窗格选择 Linearization Inspector 选项卡，就会显示每个模块的线性化信息。通过这些给定的信息确定线性化是否给出期望的结果，还可以利用不同的工作点或者参数来比较线性化结果，如图 15.30 所示。

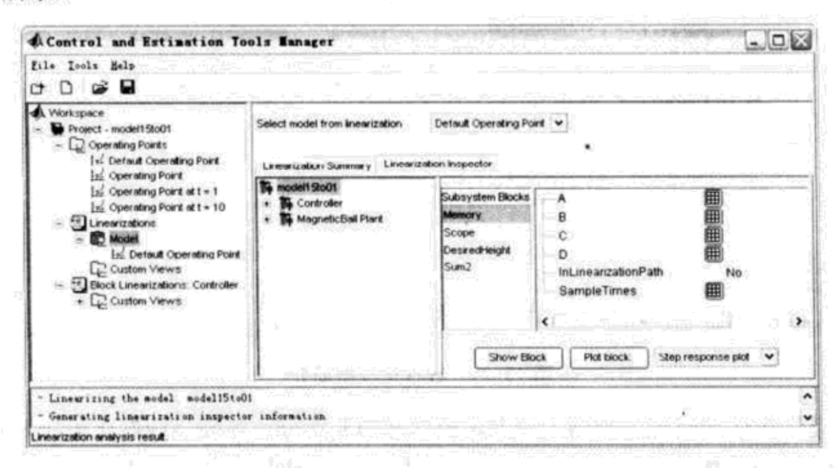


图 15.30 Simulink Control Design 的 Linearization Inspector 选项卡

注意： 当使用数值扰动线性化方法，就没有 Linearization Inspector 选项卡。

2. 显示结果

LTI Viewer 自动显示线性化结果快照，如果要创建和自定义图形，右击 Linearizations 节点下的 Custom Views 节点，在弹出的菜单中选择 Add View 命令。一个新的视图 View1 就被添加到 Custom Views 节点下，选择 Custom View1 来显示 View Setup 选项卡，如图 15.31 所示。

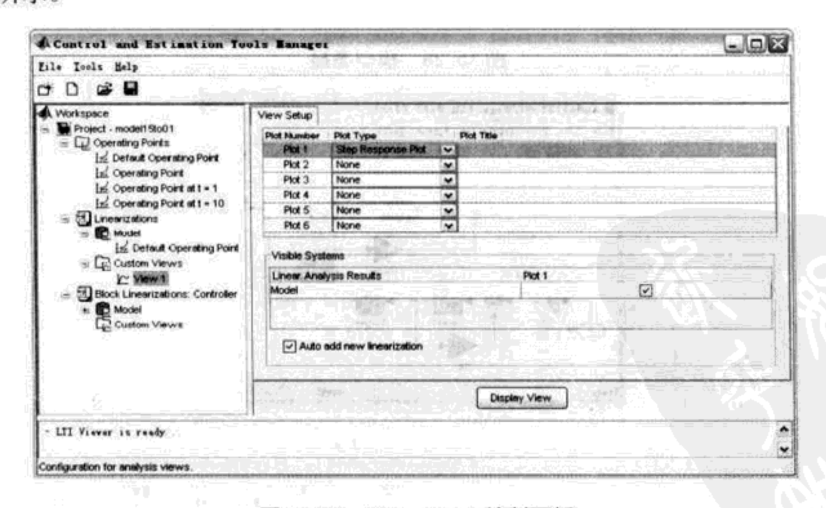


图 15.31 View Setup 控制面板

【例 1501】在配置 View Setup 控制面板之前运行另外一个在 t=10 工作点的模型线性化。

操作步骤如下：

(1) 生成 Model(2)。

具体步骤如图 15.32 所示，首先单击 Linearizations 节点，然后选择 Operating Point at t=10 选项，最后单击 Linearize Model 按钮。Model(2)就会在 Linearizations 节点下面。

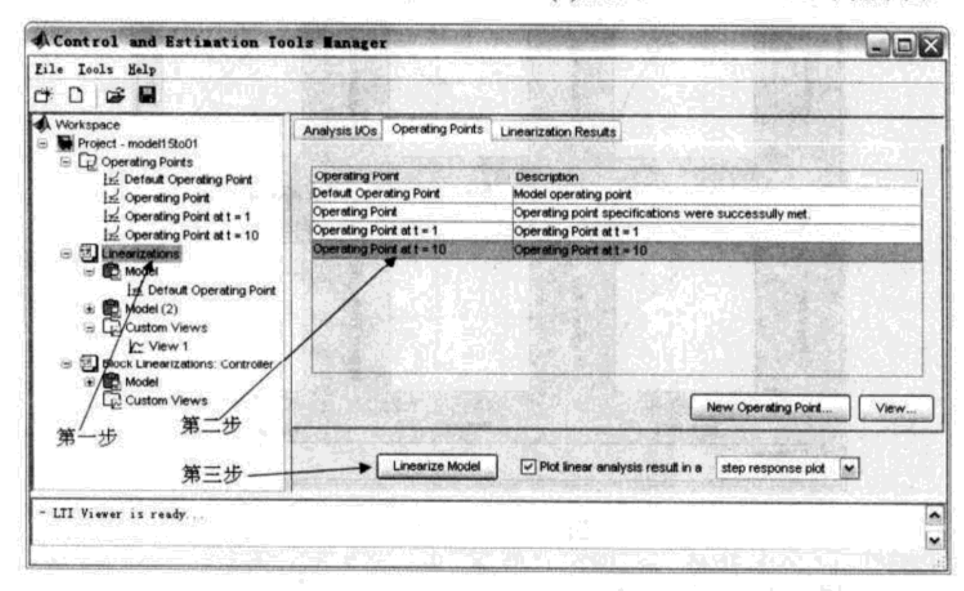


图 15.32　在 Linearizations 下生成一个新的 Model

(2) 设置 View Setup。具体设置如图 15.33 所示。

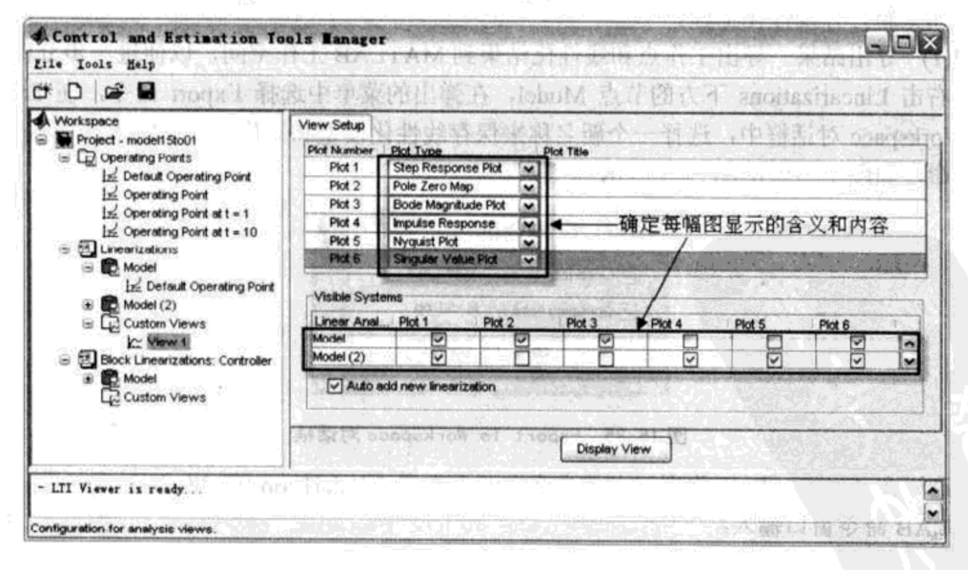


图 15.33　View Setup 设置

(3) 结果显示。

单击 Display View 来显示 LTI Viewer，如图 15.34 所示。

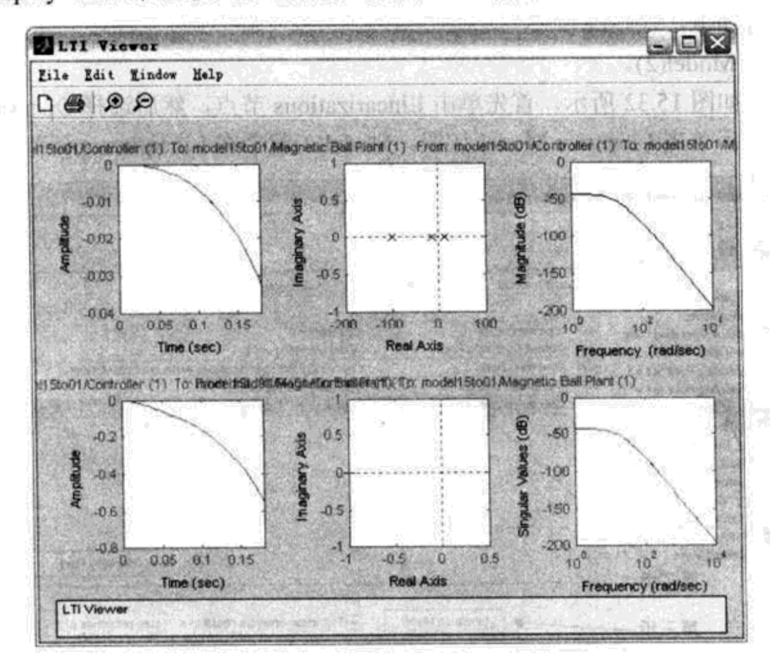


图 15.34 结果显示

15.3.10 导出并保存工程

Control and Estimation Tools Manager 窗口可以将线性化结果导入到 MATLAB 空间，保存和重新打开线性化工程。

(1) 导出结果。导出工作点和线性化结果到 MATLAB 工作空间，以便进一步分析。

右击 Linearizations 下方的节点 Model，在弹出的菜单中选择 Export 命令，在 Export To Workspace 对话框中，选择一个新名称来保存线性化模型和工作点，也可以接受默认的文件命，如图 15.35 所示，单击 OK 按钮。

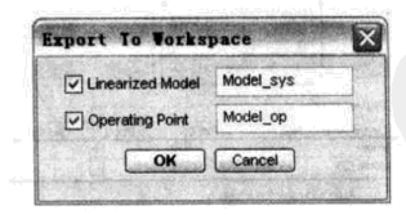


图 15.35 Export To Workspace 对话框

这样在 MATLAB 工作空间中就会多出两个变量 Model1_op 和 Model1_sys，可以在 MATLAB 命令窗口输入：

```
>> who
```

结果如下：

```
Your variables are:
L         Model_sys  beta      m
Model_op   R          g
```

还有其他方法可以导出数据。在 LTI Viewer 窗口选择 File | Export 命令，或者单击 Model 节点的 Linearization Summary 选项卡下方的 Export to Workspace 按钮。

(2) 导出和恢复线性化 I/O 设置。为了导出线性化 I/O 设置到 MATLAB 工作空间，可以使用 getlinio 函数，使用 save 函数保存这些设置，通过 load 函数来载入这些设置，利用 setlinio 函数来载入这些设置到模型当中。

(3) 保存 Control and Estimation Tools Manager 工程。一个 Control and Estimation Tools Manager 工程可能包含多个任务，包括 Simulink Control Design，Simulink Parameter Estimation 和 Model Predictive Control Toolbox 中的任务，每一个任务都含有数据，对象和特定模型的分析结果。

在 Control and Estimation Tools Manager 窗口中选择 File | Save 命令来保存工程，如图 15.36 所示。

(4) 打开 Control and Estimation Tools Manager 工程。在 Control and Estimation Tools Manager 窗口中选择 File | Load 命令来加载工程，如图 15.37 所示。



图 15.36　保存工程



图 15.37　加载工程

第 16 章　Stateflow 原理与应用

Stateflow 是有限状态机的图形实现工具，它可以用于解决复杂的监控逻辑问题，用户可以用图形化的工具来实现各个状态之间的转换。可以在 Simulink 中直接嵌入 Stateflow，达到两者的无缝连接。本章讲述了如何运行 Stateflow，如何生成 C 代码，如何利用 Stateflow 进行控制，利用 Stateflow 进行图标仿真，介绍了 Stateflow 的常用命令。按通常的惯例，最后以一个实例来说明 Stateflow 的使用步骤和基本方法。

本章主要内容包括：

- 关于 Stateflow
- 运行 Stateflow
- 为目标生成 C 代码
- 利用状态(States)和迁移(Transitions)进行控制
- 进行 Stateflow 图表仿真
- Stateflow 常用命令
- Stateflow 仿真实例

16.1　关于 Stateflow

这一节介绍 Stateflow 的最基本的概念，这些也是真正地运行一个 Stateflow 实例之前所必须具备的知识。

Stateflow 是有限状态机的图形实现工具，可以用于解决复杂的监控逻辑问题，用户可以用图形化的工具来实现各个状态之间的转换。可以在 Simulink 中直接嵌入 Stateflow，得到两者的无缝连接。Stateflow 和 Simulink 环境可以综合在一起进行建模、仿真和分析。Stateflow 可以通过图形环境来设计一个监视控制系统。通过有限状态机的图形化表示，通常可为一个相对复杂的控制系统的建模和仿真提供一个清晰明了的描述。能够将系统的描述和设计结合得更为紧密，更加容易设计，利于考虑各种情况，可以不断修改，直到设计满足要求为止。

Stateflow 仿真的原理是有限状态机(finite state machine)理论。有限状态机是指在系统中有可数的状态，在某些事件发生时，系统从一个状态转换到另一个状态，又称为事件驱动系统。

在利用有限状态机进行描述时，可以设定一个状态到另一个状态迁移的条件，最后组成迁移图。

可以利用 Stateflow 的图形设计方法，来创建有限的状态，并用图形的方式绘制出迁移的条件，从而构造出整个有限状态机系统。从上面的介绍可以看出，一个状态迁移到另外一个状态的变换，构成了 Stateflow 的基本元素，如图 16.1 所示。示意图中有 3 个状态，这几个状态之间的转换都有相应的条件，无论是各状态之间的转换，还是状态自身的

转换。

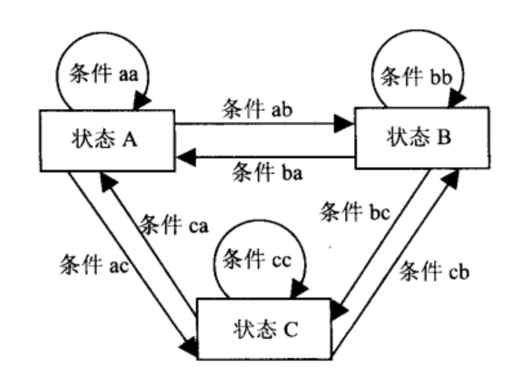


图 16.1　有限状态机示意图

16.2　运行 Stateflow

在 Simulink 中，Stateflow 模块是用 Stateflow 图形来表示一个离散模型集合的目标，这些模型就是其中的状态。通过改变控制目标状态这一事件来激发 Stateflow 有限状态机的运行。目标的行为取决于目标的状态和控制目标的状态变化。通过下面几个方面的介绍，必定会让用户对 Stateflow 如何在 Simulink 中运行有一个清晰的认识。

- Stateflow 嵌入 Simulink 中。
- 通过状态，Stateflow 来表示控制模型。
- 通过迁移，Stateflow 来改变状态。
- 通过事件来激发 Stateflow。
- Stateflow 通过连接来选择目标。
- Stateflow 使用数据变量。

16.2.1　Stateflow 嵌入 Simulink 中

Stateflow 是 Simulink 中的一种工具，能够用来表示 Simulink 建模实际系统中对控制目标的动态控制。这个控制目标可以是发动机、水泵等控制系统运行的驱动设备。在 Stateflow 图形中，通过传感器和转换器可以直观地对实际控制目标的变化进行建模。这些变化决定了 Simulink 模型行为的变化。

下面描述的是 Stateflow 演示 sf_boiler 中的一个 Stateflow 模块，通过在 MATLAB 命令窗口中输入：

```
>> sf_boiler
```

执行命令后会弹出 sf_boiler 模型，如图 16.2 所示。在模型中，Stateflow 模块控制一个由 Simulink 子系统 Boiler Plant Model 建模的锅炉。如果双击其中的 Stateflow 模块，Stateflow 图形就会弹出来，如图 16.3 所示。

单击图 16.2 或者图 16.3 中的运行按钮 ▶，可得到结果如图 16.4 所示。在仿真过程

中，图 16.3 会闪动，仿真完成后，图 16.2 会变为彩色。

用户可以先设计 Stateflow，然后设计 Simulink 模型。也可以先设计一个 Simulink 模型，然后向其中添加 Stateflow 图形。如果用户用 Stateflow 图形取代 Simulink 的逻辑模块，将会提高用户的模型性能。

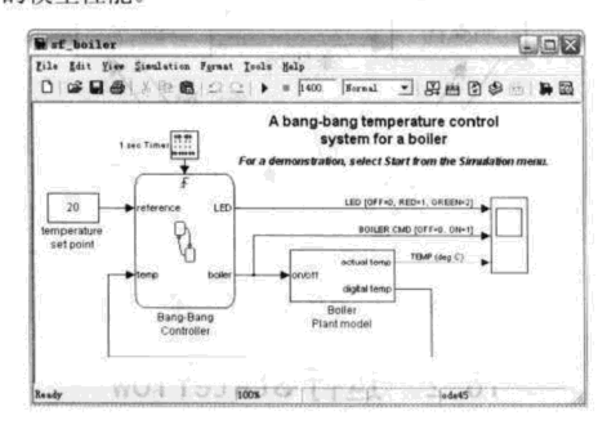


图 16.2 sf_boiler 模型

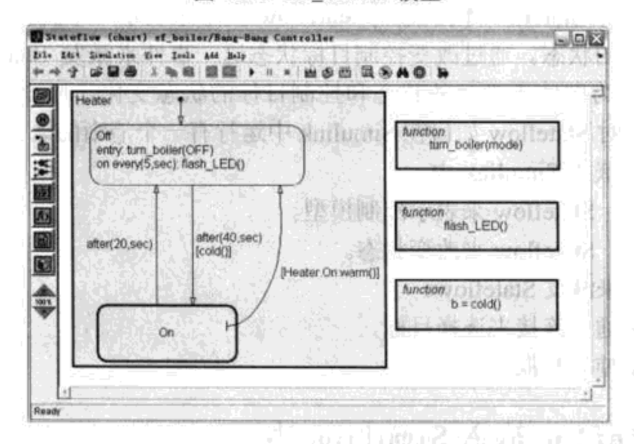


图 16.3 sf_boiler 模型的 Bang-Bang 控制 Stateflow 窗口

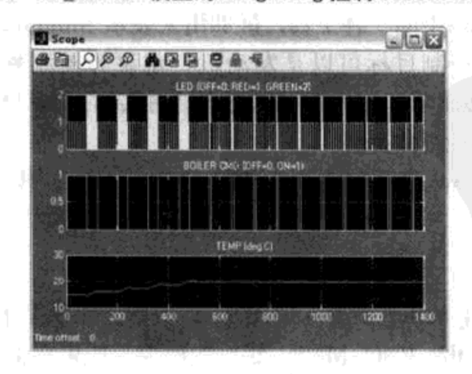


图 16.4 仿真运行结果

16.2.2 通过 Stateflow 来表示控制模型

在 MATLAB 的命令窗口中输入 stateflow 命令，打开如图 16.5 所示的窗口，其中 Chart 为空白的 Stateflow 模块。也可以直接在 Simulink 中嵌入 Stateflow 模块。

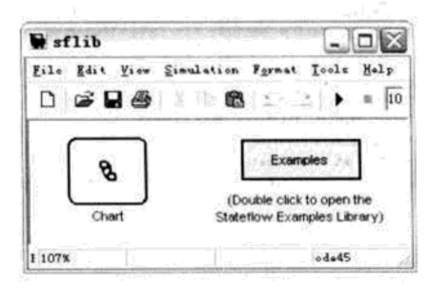


图 16.5 Stateflow 启动图

在 Stateflow 编辑界面中双击 Chart 模块，将得到如图 16.6 所示的窗口。用户可以在此窗口中编辑所需要的 Stateflow 模型。Stateflow 提供了强大的框图编辑功能，可以描述非常复杂的逻辑模型。Stateflow 框图的输出为状态，且这些状态的值是可枚举的。

在 Stateflow 编辑界面内右击，则弹出如图 16.7 所示浮动菜单，选择其中 Properties(属性)菜单，则弹出如图 16.8 所示对话框，用户可以从中设置整个模型的属性。

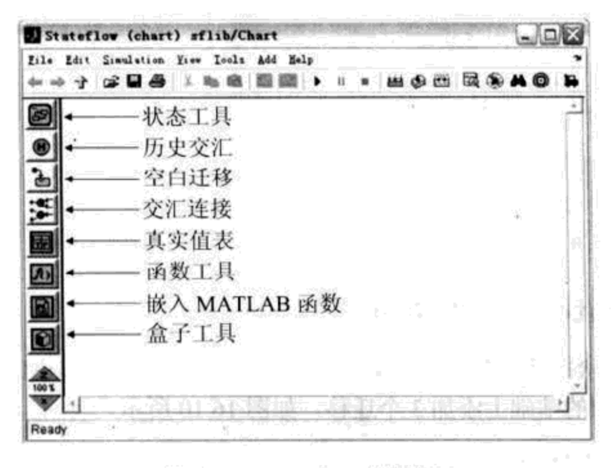


图 16.6 Stateflow 编辑窗口

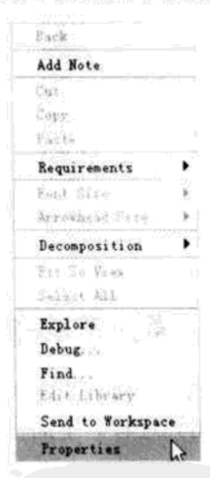


图 16.7 快捷菜单

系统的状态就是系统运行的模态。在 Stateflow 下，状态有两种行为，即“激活”(active)和“非激活”(inactive)，或称为编辑或非编辑状态。单击状态按钮，可以在图形编辑框窗口绘制一个状态的示意模块。当添加好之后，状态的示意模块就会出现一个“？”，这表示允许用户填写状态的名称和动作描述，如：on 或者 off 等。

如图 16.9 所示，有两个状态：on 和 off。右击状态图标，选择弹出的菜单中的 Properties 选项，在弹出对话框的 Label 文本框输入模块显示的名称。

当状态为 active 时，便可编写相应的执行程序，且在激活状态下可被执行。例如：如果描述风扇的状态是 on 激活的，这就表示风扇是开的。如果交通工具的 Driver 状态是激

活的，这就表示交通工具正在向前加速。

Chart Chart
Name: Chart Chart
Parent: sflib
Machine: (machine) sflib
Update method: Inherited Sample Time:
Enable C-bit operations Apply to all charts in machine now
No Code Generation for Custom Target
Export Chart Level Graphical Functions (Make Global)
Use Strong Data Typing with Simulink I/O
Execute (enter) Chart At Initialization
Debugger breakpoint: On chart entry Editor: Locked
Description:
Document Link:
OK Cancel Help Apply

图 16.8 属性设置对话框

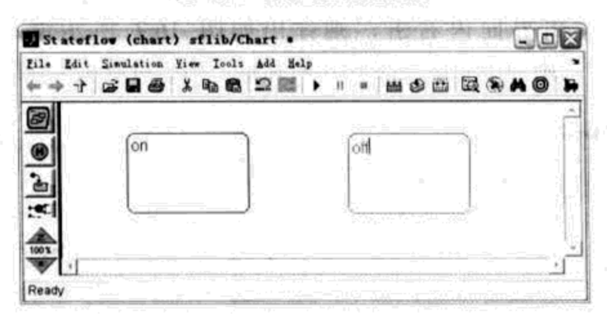


图 16.9 Stateflow 窗口

16.2.3 通过迁移来改变 Stateflow 状态

仅通过状态还不足以对变换激活状态的目标来进行建模。为激活状态的变换提供路径，还要用到迁移。下面在上一节实例的基础上添加 3 个迁移，如图 16.10 所示。

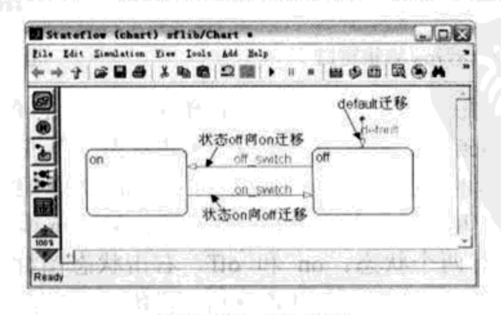


图 16.10 添加迁移

状态迁移指在一个状态的边界单击鼠标，并拖动到另外一个状态处释放，则可以绘制出从一个状态转换成另外一个状态的连线。

右击 on-off 迁移状态线，则将会弹出一个窗口，其中空白迁移则需要单击窗口左边的空白迁移按钮来绘制。

要特别注意迁移的方向，即起始于源状态，终止于目标状态。若源状态被激活后，执行了迁移，那么源状态就会变成非激活状态，而目标状态会被激活。例如，off 处于激活状态，则通过 off-on 的迁移，on 的状态就会被激活。但若 on 处于激活状态，却不能通过 off-on 迁移来使 off 状态被激活。如果要实现这个目标，就必须采用 on-off 迁移。

其中还有一类比较特殊的迁移，就是空白迁移。这种迁移指向 off 状态，但是没有明显源状态。空白迁移用来表示两种状态中的一种，即 on 和 off，当 Stateflow 图形为激活时也随之为激活。

16.2.4　通过事件来激发 Stateflow

事件为模型中状态间的转换提供了驱动。一个事件可以是一个温度点，表示房子内的温度超过给定的温度。事件也可以是一个水准点，当容器中的水过多时，就会自动溢出。通常，利用 Simulink 中建立的实际模型来产生这些事件，然后触发这些 Stateflow 模块作出相应的反应。

要记住的一件重要的事情就是，只有当事件发生时，Stateflow 图形才会变为激活的。同样如果没有事件的驱动，状态间的迁移也不会发生。前面的实例中，off 状态是激活的，如果没有收到更新的事件，off 状态仍然是激活的。

事件可以在 Stateflow 中表示，图形中可通过事件触发器来查看事件，如图 16.11 所示。

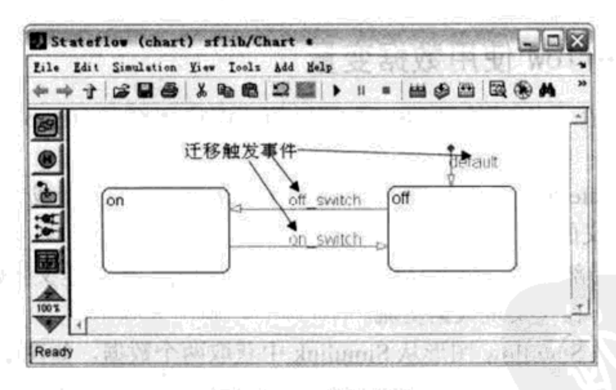


图 16.11　触发事件

在这个实例中，当且仅当 off 处于激活状态，并且收到 on_switch 事件，才能实现由状态 off 到状态 on 的迁移发生。同样，当且仅当 on 处于激活状态，并且收到 off_switch 事件，才能实现由状态 on 到状态 off 的迁移。

16.2.5　Stateflow 通过连接来选择目标

状态、迁移和事件是状态的基本模块，此外 Stateflow 还提供了连接来表示迁移中的决策点。连接为迁移提供了选择的路径。

在上面实例的基础上添加一个连接，如图 16.12 所示。

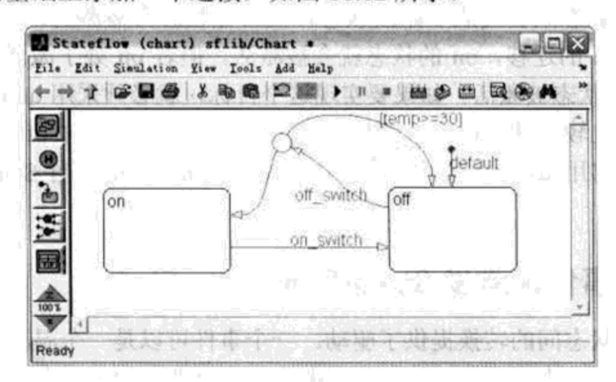


图 16.12　目标选择

如果状态 off 迁移到连接，那么随后连接输出端的其中一个迁移就会发生。在这个例子中，如果温度大于或等于 30℃时，状态 off 就会重新返回状态 off，这样状态 off 仍然处于激活状态。如果是小于 30℃，那么状态 off 就会便被迁移到状态 on。自此，状态 on 处于激活状态，状态 off 处于非激活状态。

一个决策逻辑可以通过流程图符号来创建，如 for 循环和 if-then-else 结构。在很多情况下，使用流程图符号能够更加确切地表示所需要的系统，并可以避免不必要的状态。

16.2.6　Stateflow 使用数据变量

下面的实例是在上面讨论过的，其中外面的温度用变量 temp 表示，如图 16.12 所示。

temp 是一个 Stateflow 数据的例子。Stateflow 数据可以是变量或者是常数，这些都是为 Stateflow 图形定义的。

可以定义一个局部变量，仅在 Stateflow 图形中使用，或者定义各输入或输出数据，从而可以与 Simulink 模型实现数据交流。

下面这个实例是 Stateflow 图形从 Simulink 中获取两个数据，如图 16.2 所示。

这个例子中，变量 reference 是从 Simulink 中获取一个常量，作为参考点。变量 temp 作为 Stateflow 的输入，来自于 Simulink 仿真中不断更新的系统实际温度。

16.3　为目标生成 C 代码

完成建模后，可以生成模型中控制部分的代码，以充当实际系统中的实际控制器。这样用户就可以使用 Stateflow 的代码生成器。Stateflow 可以为指定的目标产生代码，这些

目标就是 Stateflow 机中的对象。Stateflow 机即包含 Simulink 模型中所有 Stateflow 模块的容器。目标对象函数作为一个容器来产生 Stateflow 机中 Stateflow 模块代码。对于 Stateflow 机有 3 个类型的目标。

- 仿真目标(Simulation Targets)

如果使用 Stateflow 进行仿真时，Stateflow 为每一个 Stateflow 机产生一个 S 函数(MEX-文件)，用来支持 Stateflow 图模型仿真。在 Stateflow 中的仿真目标的名称为 sfun。

Stateflow 利用产生的代码来对图进行仿真，因为 Stateflow 在进行 Stateflow 图仿真时，都会自动产生 C-代码用来仿真。当产生仿真代码时，需要参考仿真目标 sfun。

- Real-Time Workshop 目标(Real-Time Workshop Targets)

Real-Time Workshop 能够从由 Stateflow 代码生成的 Simulink 建立能够嵌入目标的代码，关系如图 16.13 所示。

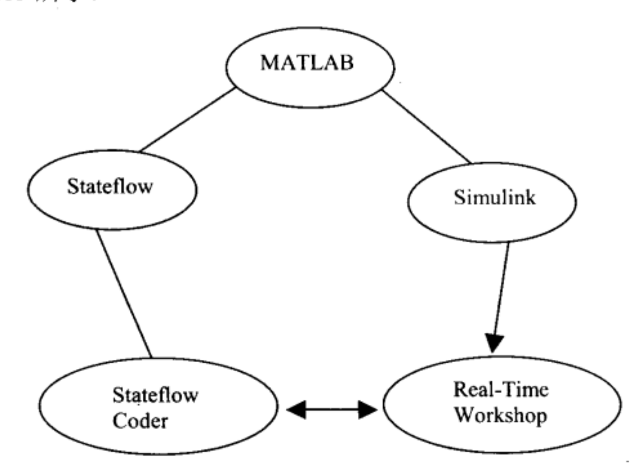


图 16.13 Stateflow 与 Simulink 关系图

Stateflow 代码生成器产生整数还是浮点型代码，取决于 Stateflow 机。Real-Time Workshop 从 Simulink 中模型的端口产生 Stateflow 代码，并可为实时运行 Stateflow 提供一个框架。由 Stateflow 代码产生器产生的代码能够无缝地嵌入到 Real-Time Workshop 产生的代码中，可能想设计一个特定的方案，使得来自 Stateflow 和 Real-Time Workshop 两个产品的代码能够为一个特定的平台所使用。如果有 Real-Time Workshop 工具的话，可以从这两个产品中获取代码，并可作为一个应用程序在其他环境中使用，来控制一个过程。

- 自定义目标(Custom Targets)

如果用户有 Stateflow 代码器，可以为在其他环境中编写的目标软件产生代码。使用 Stateflow 代码器来生成自定义目标，这些都是专门为 Simulink 模型中的 Stateflow 机端口所设置的，自定义目标也有一个独一无二的名称，就如同 Stateflow 中的 sfun,rtw。

16.4 利用状态和迁移进行控制

这一节通过制作和执行一个简单的控制模型来进一步理解 Stateflow。这个模型由一个简单的 Stateflow 模块和一些相关的 Simulink 模块组成。这个 Stateflow 模块就是一个简单的 Stateflow 图。在进行后面的章节前，首先还要对 Stateflow 图和 Simulink 模块有一个充分的了解。

1. 创建一个 on-off 模型

这一部分学习如何使用 Stateflow。在 Simulink 中创建一个 Stateflow 模块，用来表示 on-off 控制，这个模型是提供进一步学习的基础。随着不断地改善下面实例中的模型，就会逐渐了解如何在 Simulink 中使用 Stateflow 了。

创建一个带有 on-off 控制的控制模型，操作步骤如下：

(1) 创建一个带有 Stateflow 模块的 Simulink 模型。在 Simulink 中，通过一些模块来创建实际系统的模型。Stateflow 模块就是 Simulink 模型的一部分，可以在 Simulink 模型中使用 Stateflow 模块来控制一个实际系统对象，如发动机、水泵或者是风扇。

本部分就开始建造一个带有 Stateflow 的 on-off 控制的实例模型。通过以下几个步骤来创建并保存一个带有一个 Stateflow 模块的 Simulink 模型。

① 在 MATLAB 命令窗口中输入 sfnew 命令，就会创建一个带有 Stateflow 模块的 Simulink 模型，Simulink 模型文件名为 untitled，Stateflow 名为 Chart，如图 16.14 所示。

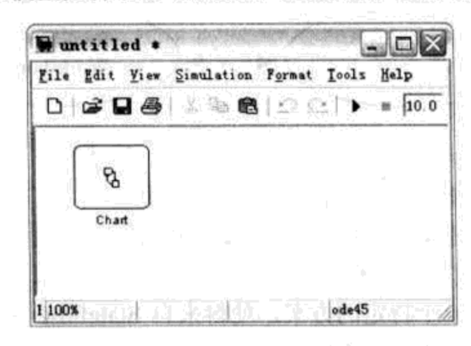


图 16.14 带有 Stateflow 模块的模型

② 在 Simulink 窗口中，单击 Stateflow 模块下方的 Chart，模块下方的标签变为可编辑状态。

③ 将 Chart 修改为 Controller，可以暂时称 Simulink 中的 Stateflow 模块为“控制器 Stateflow 模块”或者仅仅为“控制器模块”。

(2) 保存 Simulink 模型。保存 16.3 节创建的模型，选择 Simulink 窗口中的 File | Save as 命令，选择想要保存的路径，然后保存文件名为 model16to01.mdl。

(3) 打开 Stateflow 模块图表，可以创建 Stateflow 模块来编写 Simulink 模型实际系统的控制目标。例如，可以通过 Stateflow 模块来断定系统压力和温度是否太高，然后通过调节来降低系统的压力和温度。这些都可以通过编辑 Stateflow 图表来获取 Simulink 中的

Stateflow 模块。

打开 Stateflow 图表的方法是：在 Simulink 窗口中双击 Controller 模块，弹出一个空白的 Stateflow 图表编辑窗口。在打开的窗口中，必须注意以下几点：

- 进行图表编辑的 Stateflow 模块名出现在标题栏中。
- 左边的标题栏是用来绘制组成 Stateflow 图表的不同图形目标。要特别注意绘制 Controller 模块 Stateflow 图表的绘图工具：状态工具(State tool)和默认迁移工具(Default Transition tool)。

(4) 在 Stateflow 图表中绘制状态

在 Stateflow 图表中为 Stateflow 模块编写控制行为。状态就是 Stateflow 图表所要表达的目标，它代表控制的各种操作模式。本例中，控制的状态模式只有两种：on 和 off。首先绘制 on 状态和 off 状态，具体步骤如下：

① 在 Stateflow 图表窗口左边，单击状态工具。

② 将鼠标移动到可编辑区，此时鼠标周围就会出现一个带圆角的方框。

③ 在可编辑区的左上角单击，如图 16.15 所示，一个高亮度的状态就会出现，且其左上角有一个闪动的文本光标。

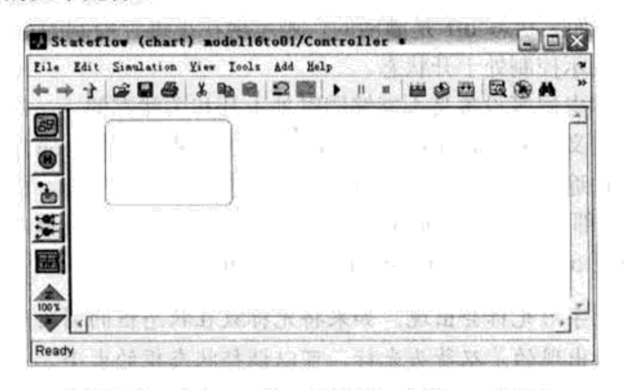


图 16.15　在 Stateflow 编辑窗口中添加一个状态

④ 在状态中闪动文本光标处输入 on，就为这个新的状态取名为 on，如图 16.16 所示。

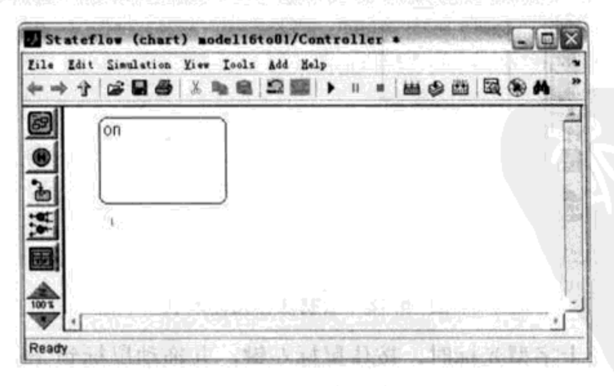


图 16.16　状态添加文本说明

⑤ 按前面同样的过程，创建一个新状态，取名为 off，如图 16.17 所示。

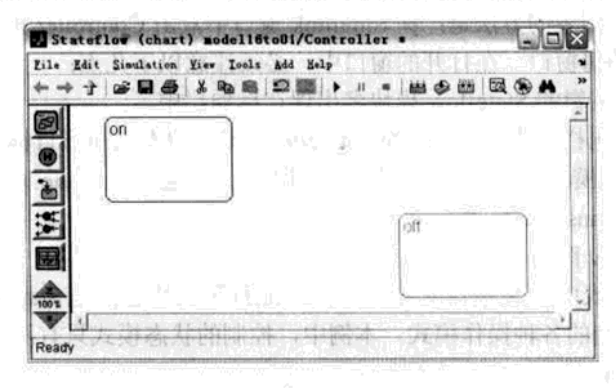


图 16.17 创建状态 on 和状态 off

Stateflow 图表是一种有限状态机，有限状态机用各种状态来表示目标的可能控制模式。上面创建了两个状态，分别为 on 和 off。这两种状态分别表示两种控制模式：开或者关。

(5) 在 Stateflow 图表中绘制迁移。Stateflow 图表中的状态可以是激活的(active)或是非激活的(inactive)。如果状态 off 是激活的，表示现在正进行控制的是关闭状态，如果 on 状态时激活的，则表示控制处于开状态。

由于像发动机和电扇等设备是不能够同时处于开和关状态的，一次只能有一个状态处于激活。图表必须定义控制状态变化的方式，以便达到从 off 转向 on 和 on 转向 off。图表中的迁移定义了一种途径，通过它，可以是一个状态变为激活状态，而另外一个状态转变为非激活状态。这一部分中，将在 off 状态和 on 状态之间建立一个迁移，具体步骤如下：

① 将光标移至 off 状态方框的左边界，如图 16.18 所示。

注意： 此时十字形光标会出现。如果将光标放在状态框的圆角处，十字形光标就会消失，出现的是双箭头光标，可以调整状态框的大小。

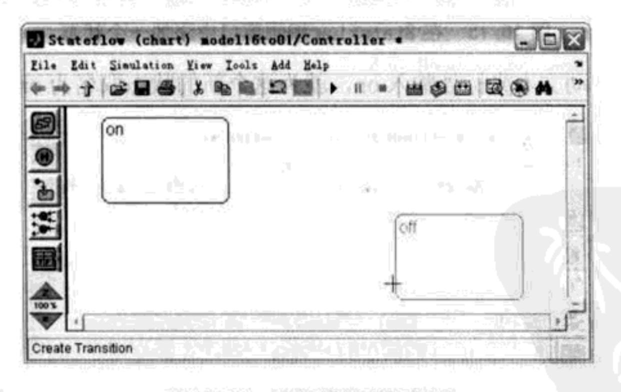


图 16.18 绘制迁移光标形状

② 当鼠标变成十字型光标时，按住鼠标左键，并拖动鼠标到 on 状态框的下边框。带有迁移的 Stateflow 图表如图 16.19 所示。

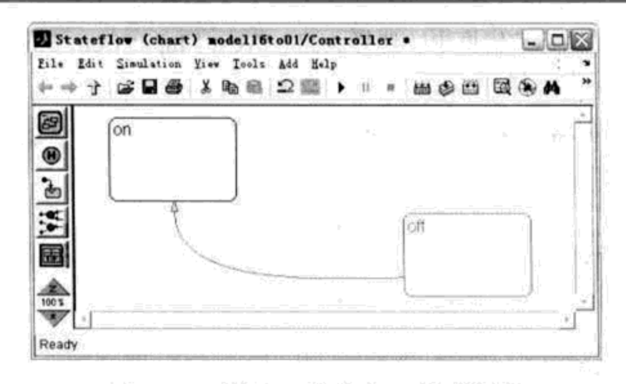


图 16.19　绘制 off 状态到 on 状态的迁移

③　按照上一步同样的过程，从 on 状态右边到 off 状态顶边绘制一个迁移。

迁移是由状态 on 至状态 off 的，这意味着如果 on 状态是激活的，那么通过迁移，off 状态成为激活的，而 on 状态则变为非激活的。

还必须添加一个最终迁移，用来告诉 Stateflow 图表其运行的状态，即处于 off 或者 on 状态。可以使用一个空白(default)迁移，使得 off 状态作为最开始的激活状态，具体可见后面的步骤。

④　添加空白(Default)迁移图标。单击工具栏空白(Default)迁移图标，将光标移动到绘图区，将光标移动到 off 状态框上边界的右边。当光标的箭头接触到 off 状态框的上边界，则立即会竖立起来，并且和状态框连成一体，松开鼠标，如图 16.20 所示。

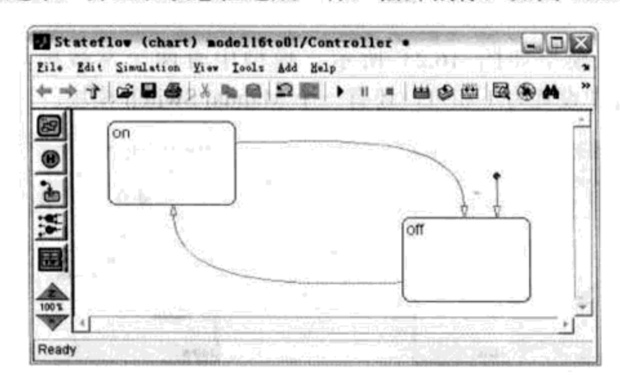


图 16.20　迁移添加完成后的文件窗口

(6)　为 Stateflow 图表添加一个触发事件。

当 Stateflow 图表接收到 Simulink 输入的触发事件后就会执行。在 Stateflow 图表中绘制迁移时，就为 on 状态和 off 状态在激活或者非激活之间提供了迁移。现在要做的事情就是为 Stateflow 的图表的执行提供触发事件。这里就为上面这个实例定义一个触发事件。还有另外一种方法发送事件来更新 Stateflow 图表。例如，可以让 Stateflow 图表在每次 Simulink 样本更新的时候执行。然而，触发事件却可以使 Stateflow 图表以最大的独立性来进行独立的控制。

通过以下步骤来定义一个更新 Controller 图表的触发事件：

① 在 Stateflow 图标编辑框中，选择 Add | Event | Input from Simulink 命令，弹出新事件的属性对话框如图 16.21 所示。

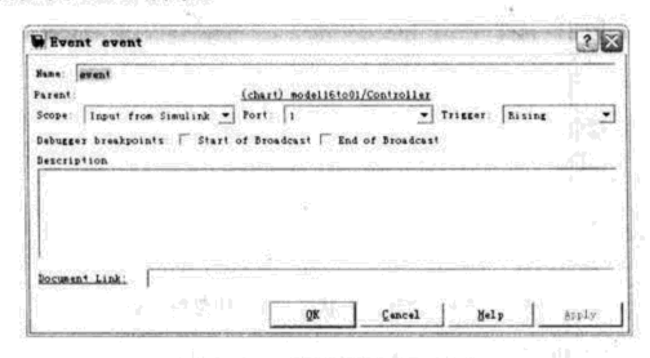


图 16.21　事件属性设置对话框

② 选择其中的 Trigger 选项为 Either。

③ 单击 OK 按钮，改变和关闭属性对话框，如图 16.22 所示。

(7) 向 Stateflow 图表传递一个触发事件。

先来定义一个触发事件。Simulink 中的 Controller 模块有一个触发端口。为了更新 Controller 图表，Simulink 模型必须向触发端口发送一个触发信号。

通过开始定义的 Either 为 Controller 图表定义一个触发事件。当输入触发端口的控制信号过零上升和下降，事件就会发生。下面，就是用一个手工的切换来提供一个上升或下降的控制信号。

建立 Simulink 模型如图 16.23 所示。模型中有两个常数模块 Constant 模块和 Constant1 模块，分别设置为 1 和-1，还有一个 Manual Switch 模块。

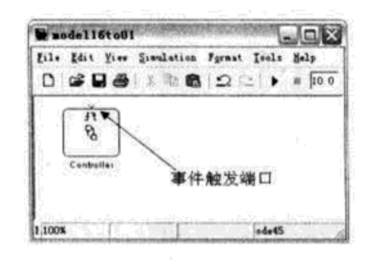


图 16.22　增加触发事件的系统

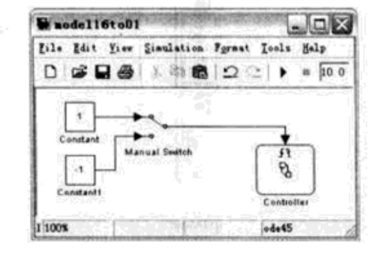


图 16.23　Simulink 模型

16.5　进行 Stateflow 图表仿真

在上一节建立好 Simulink 模型之后，这里来测试一下。在仿真过程中，Stateflow 图表会产生一个图形来显示迁移的执行和当前的激活状态。通过这个，可以发现 Stateflow 图表是否是按所期望的方式执行。如果需要，可进行适当的切换。

以下各节将对模型中的 Controller Stateflow 图表进行仿真，主要包括：

- 定义模型仿真参数。

- Stateflow 图表仿真的基本步骤。
- 仿真过程中的调试。

16.5.1　定义模型仿真参数

默认可以通过 Simuink 模型和 Stateflow 模型来激活仿真。为了确保能够激活 model16to01.mdl 模型，设置 Controller Stateflow 图表的仿真参数的操作步骤如下：

(1) 打开 Simulink 模型仿真参数对话框，选择 Simulation | Configuration Parameters 命令。

(2) 在 Stop time 栏中，输入 inf，然后单击 OK 按钮。

将停止时间设置为 inf，表示 Simulink 的仿真时间为无穷大，直到用户让它停止为止。

(3) 在 Stateflow 编辑框，选择 Tools | Open Simulation Target 命令，弹出 Simulation Target Builder 参数对话框如图 16.24 所示。

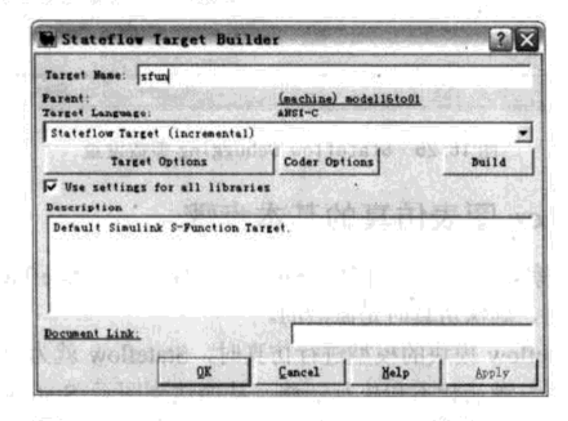


图 16.24　Simulation Target Builder 参数对话框

(4) 在 Simulation Target Builder 对话框中单击 Coder Options 按钮，弹出 Coder Options 对话框。

(5) 选中 Enable debugging/animation 复选框，如图 16.25 所示，单击 OK 按钮应用并关闭参数对话框。

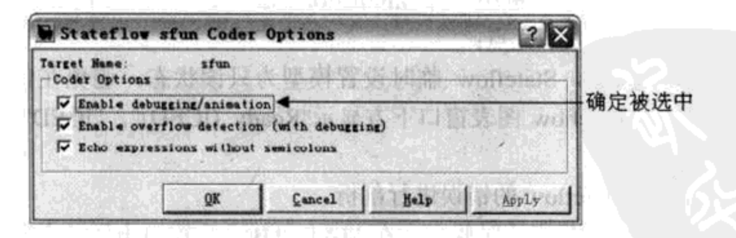


图 16.25　Coder Options 参数对话框

(6) 返回到 Simulation Target Builder 对话框，单击 OK 按钮。

(7) 在 Stateflow 编辑框，选择 Tools | Debug 命令。

(8) 在 Animation 内，选中 Enabled 单选按钮，Delay (sec)文本框设置为 0.6，如图 16.26

所示。在 Delay (sec)栏输入一个较大的值可以降低仿真的速度，输入较小的值可以加快仿真速度。

(9) 单击 Close 按钮，关闭 Stateflow Debugging 参数设置对话框。

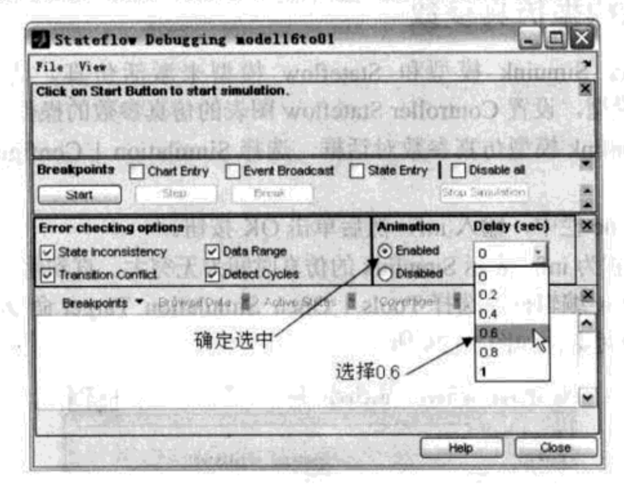


图 16.26 Stateflow Debugging 参数设置

16.5.2 Stateflow 图表仿真的基本步骤

Stateflow 允许观察模型仿真过程中 Simulink 模型中 Stateflow 图表的行为。在 Stateflow 图表仿真之前，确保仿真时是激活的。

当对一个含有 Stateflow 模块的模型进行仿真时，Stateflow 就为每一个 Stateflow 模块创立一个应用程序来亮显激活状态和执行迁移，从而达到图示 Stateflow 图表的运行。在进行 Stateflow 图表仿真时，必须知道用它来是做什么的，而且要注意它的行为是否是所期望的。

对 model16to01.mdl 模型的仿真，操作步骤如下：

(1) 在进行 model16to01.mdl 模型仿真之前，对 Simulink 模型窗口和 Controller Stateflow 图表窗口进行位置调整，以便在仿真过程中能够同时观察到这两个窗口。

(2) 在 Stateflow 图表编辑框中选择 Simulation | start 命令，开始模型仿真。

仿真过程中需要注意以下几点：

- 在进行仿真之前，Stateflow 临时设置模型为只读状态，避免在仿真过程中被修改。将会在 Stateflow 图表窗口下方显示"Ready (ICED)"，"ICED"是 Stateflow 的内部标示符。
- 对 Controller Stateflow 的错误进行剖析。
- 在 Stateflow 进行仿真的时候，会在 MATLAB 命令窗口出现仿真的相关信息。

(3) 在 Stateflow 图表仿真开始时，背景变暗，在仿真过程中，Controller 图表处于激活状态。

(4) 在 Simulink 窗口中双击 Manual Switch 模块，双击 Manual Switch 模块可以在 1 和-1 之间相互转换。由于 Controller 模块的触发事件设置为 Either，当发生任何方向的转

换(1 到-1 的下降或者-1 到 1 的上升)时，Controller 模块都认为是触发事件。

在 Stateflow 图表执行的时候，可以发现第一个执行的迁移，就是空白(default)迁移，它迁移的目标就是 off 状态，图 16.27 为由空白迁移到 off 状态。

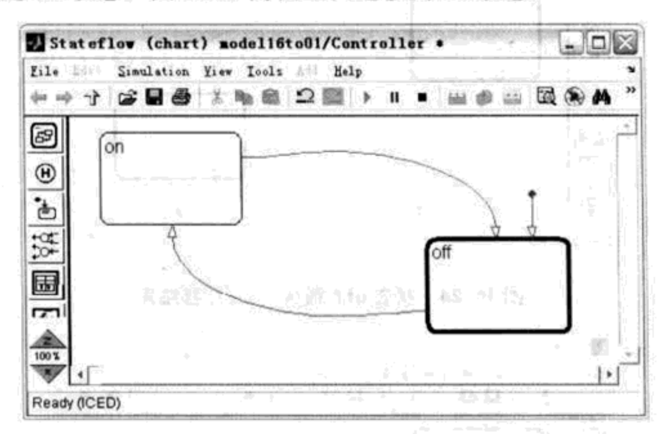


图 16.27　执行空白迁移激活 off 状态

(5) 在 Simulink 窗口再次双击 Manual Switch 模块，这就为 Stateflow 图表发送另外一个触发信号，就会执行 off 到 on 的迁移，此时迁移就会变为蓝色并加粗，随即状态 off 变为黑色，同时状态框变细，同时状态 on 变为蓝色并加粗，随即迁移变为黑色，同时迁移线变细，如图 16.28～图 16.29 所示。

(6) 在 Simulink 窗口再次双击 Manual Switch 模块，这就为 Stateflow 图表发送另外一个触发信号，就会执行 on 到 off 迁移，此时迁移就会变为蓝色并加粗，随即状态 on 变为黑色，同时状态框变细，同时状态 off 变为蓝色并加粗，随即迁移变为黑色，同时迁移线变细，如图 16.30～图 16.31 所示。

(7) 通过双击 Simulink 模型中的 Manual Switch 模块来不断地向 Stateflow 发送触发信号，而 Stateflow 中的状态也不断地重复步骤(5)和(6)。

(8) 单击 Simulink 中的 Stop 按钮可停止仿真，此时 Stateflow 又回到可编辑状态。

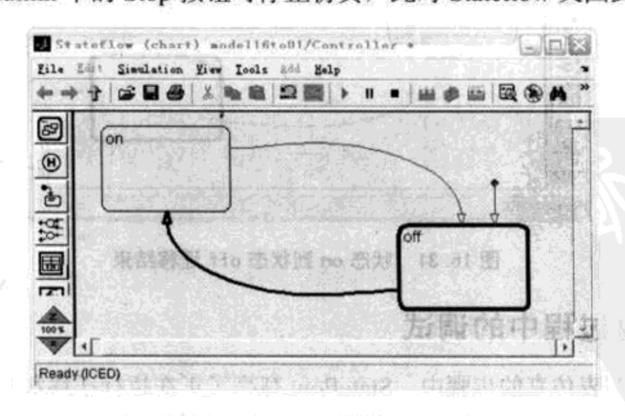


图 16.28　状态 off 到状态 on 迁移开始

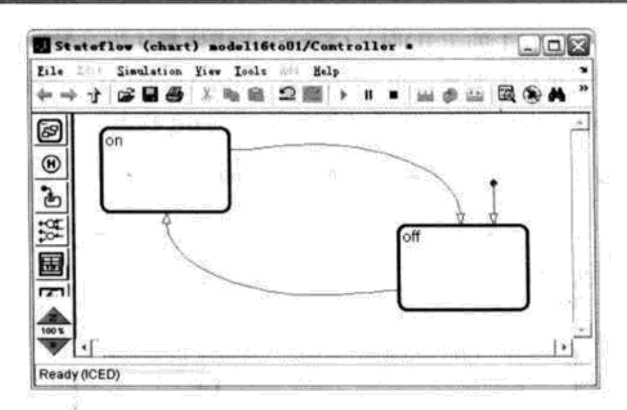


图 16.29 状态 off 到状态 on 迁移结束

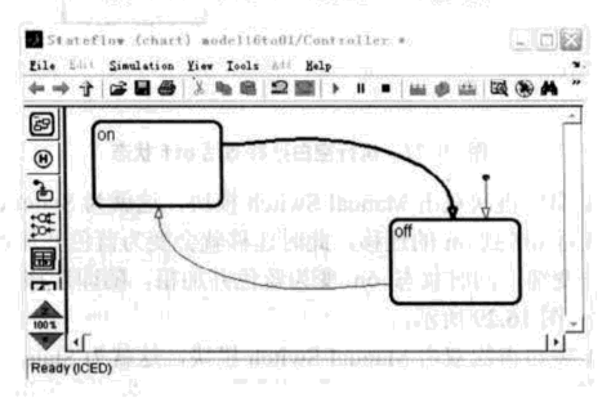


图 16.30 状态 on 到状态 off 迁移开始

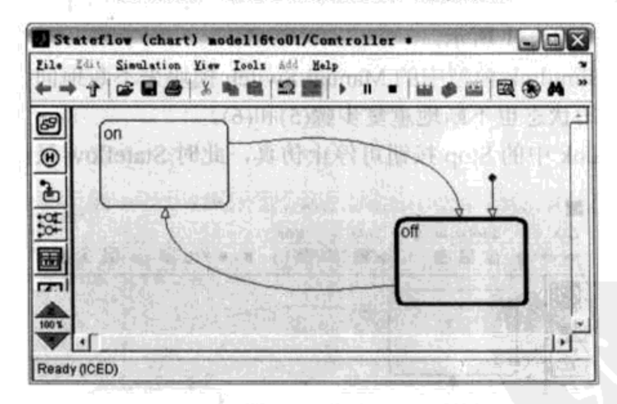


图 16.31 状态 on 到状态 off 迁移结束

16.5.3 仿真过程中的调试

在 Stateflow 图表仿真的步骤中，Stateflow 高亮了正在执行迁移和激活的状态。

可以使用 Stateflow Debugging 调试工具来增强规范性。此工具可以让用户了解每一步的 Stateflow 作用，这个在默认情况下是不具备的。通过了解 Stateflow 图表的每一个作

用，这个工具能够让用户对 Stateflow 的内部运行行为有一个更加清晰的认识。

利用 Stateflow Debugging 工具来调试 Controller Stateflow 图表仿真的，操作步骤如下：

(1) 在 Stateflow 图表编辑框中选择 Tools | Debug 命令，弹出 Stateflow Debugging 窗口。

(2) 在 Breakpoints 栏中选中 Chart Entry 复选框，如图 16.32 所示。此选项的作用是当从 Manual Switch 发送触发信号之后，进入图表暂停运行 Stateflow。

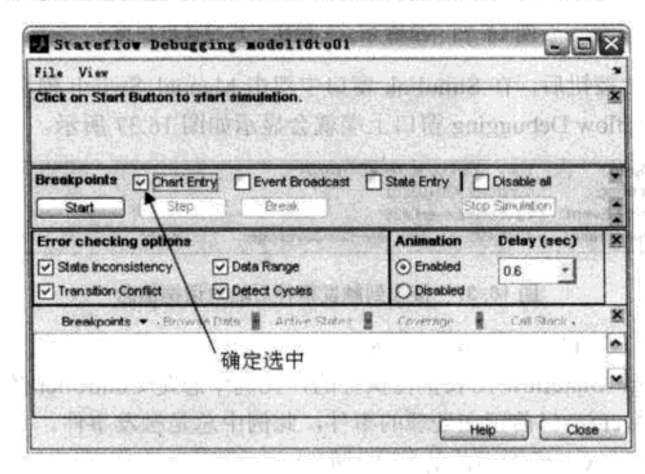


图 16.32 选择 Breakpoint 栏中 Chart Entry 选项

(3) 放好 Stateflow Debugging 窗口的位置，以便能够同时看到 Stateflow Debugging 窗口、Simulink 模型窗口和 Controller Stateflow 图表窗口。

(4) 在 Stateflow Debugging 窗口中，单击按钮 Start 按钮运行仿真，如图 16.33 所示。当仿真开始时，Start 按钮被重新命名为 Continue 按钮。但此时 Continue 和 Step 按钮均不可用，这就意味着 Controller 图表处于非激活状态，在等待事件的发生，如图 16.34 所示。

在 Stateflow Debugging 窗口上端，在单击 Start 按钮之前的信息如图 16.35 所示。在单击 Start 按钮之后的信息如图 16.36 所示，表明在等待触发事件的发生。Controller 图表虽然被激发，但是处于睡眠状态。



图 16.33 单击按钮 Start 前

图 16.34 单击按钮 Start 后

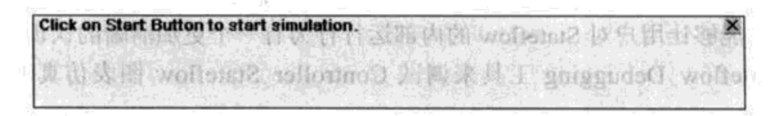


图 16.35 单击 Start 按钮之前对话框信息

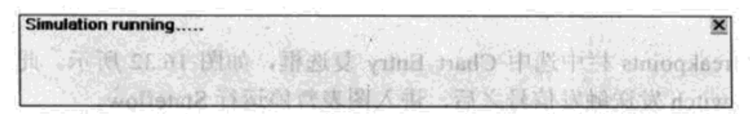


图 16.36 单击 Start 按钮之后对话框信息

(5) 单击 Start 按钮后，在 Simulink 窗口中双击 Manual Switch 模块向 Controller 图表发送事件，在 Stateflow Debugging 窗口上端就会显示如图 16.37 所示。

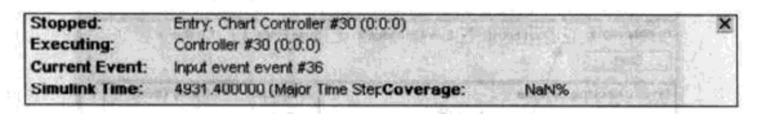


图 16.37 接收到触发事件后的对话框信息

- Stopped：当前步正在处理。
- Executing：Stateflow 图表正在执行的，此例中总是 Controller 图表。
- Current Event：目前正在处理的事件，此例中总是触发事件。
- Simulink Time：模型当前的仿真时间。

现在，先只注意 Stopped 的信息。这个信息表示当前仿真步正在处理中，如图表 16.1 所示。

表 16.1 执行过程及其示意图

执行过程	示意图
Stopped: Entry:Chart Controller #30 (0:0:0) 表示 Controller 图表处于激活状态并且准备接受触发事件时发生这一步，Stateflow 窗口如图 16.38 所示。	on off 图 16.38

(6) 在 Stateflow Debugging 对话框，单击 Step 按钮，让仿真前进一步，在 Stateflow Debugging 对话框上端就会显示如图 16.39 所示信息，解释如表 16.2 所示。

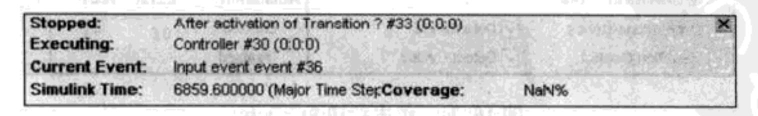
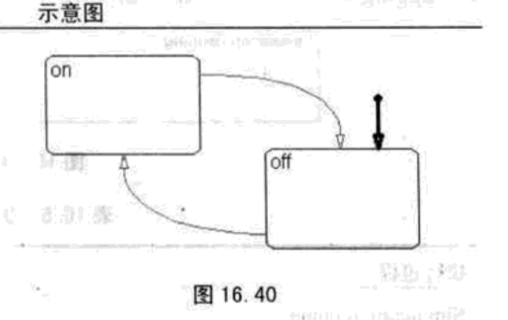


图 16.39 执行空白迁移

表 16.2　执行过程及其示意图

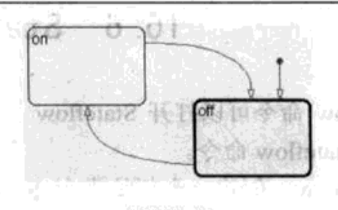
执行过程	示意图
Stopped: After activation of Transition?#33(0:0:0) 图表正在执行空白迁移，而 Debugging 窗口将引用目前迁移的标签。因为当前迁移没有标签，所以就会出现一个问号(?)	 图 16.40

(7) 在 Stateflow Debugging 窗口中，单击 Step 按钮，让仿真前进一步，在 Stateflow Debugging 窗口上端就会显示如图 16.41 所示。

Stopped:	Just after activation of State off #31 (0:0:0)
Executing:	Controller #30 (0:0:0)
Current Event:	Input event event #36
Simulink Time:	2576.000000 (Major Time Step **Coverage:** NaN%

图 16.41　Stateflow Debugging 窗口上端的显示

表 16.3　执行过程及其示意图

执行过程	示意图
Stopped: Just after activation of State off. 状态 off 处于激活	 图 16.42

(8) 在 Stateflow Debugging 窗口中，单击 Step 按钮，让仿真前进一步，在 Stateflow Debugging 窗口上端就会显示如图 16.43 所示。

Stopped:	After broadcast of Input event event #36 (0:0:0)
Executing:	Controller #30 (0:0:0)
Current Event:	Input event event #36
Simulink Time:	2659.200000 (Major Time Step **Coverage:** NaN%

图 16.43　触发事件发送后的显示

表 16.4　执行过程及其示意图

执行过程	示意图
Stopped: After broadcast of input event event. 触发事件发送后 Stateflow 图表的反应，然后事件无用	Stateflow(Chart)窗口状态如图 16.42 所示，此处不再重复

(9) 在 Stateflow Debugging 窗口中，单击 Step 按钮，让仿真前进一步。在 Stateflow Debugging 窗口上端就会显示如图 16.44 所示。

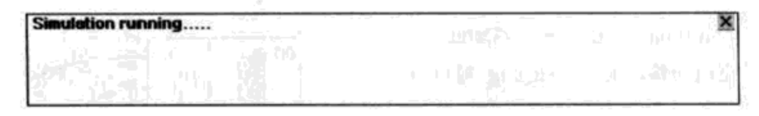


图 16.44 等待触发事件

表 16.5 执行过程及其示意图

执行过程	示意图
Simulation running.... Controller 图表被激活，但是处于睡眠状态。此时等待接受触发事件来唤醒 Controller 图表。此时 Continue 按钮和 Step 按钮处于不可用状态，直到有触发事件发生	Stateflow(Chart)窗口状态如图 16.42 所示，此处不再重复

(10) 在 Simulink 模型中，连续地双击 Manual Switch 模块来发送事件给 Controller 图表，然后单击 Step 按钮让仿真前进一步。

(11) 当调试完成，可单击 Stop Simulation 按钮结束仿真，一旦仿真停止，模型就变为可编辑状态。

(12) 在 Stateflow Debugging 窗口，单击 Stop 按钮，关闭窗口。

16.6 Stateflow 常用命令

stateflow 命令可以打开 Stateflow 主界面，启动 Stateflow 编辑环境。除此之外，另外介绍几个 stateflow 命令。

- sfnew：创建一个新的带有 Stateflow 模块的 Simulink 模型，使用"sfnew 模型名"可以直接以指定的文件名创建一个新的带有 Stateflow 模块的 Simulink 模型。
- sfexit：关闭包含 Stateflow 的模型窗口，并关闭 Stateflow 编辑窗口。
- sfsave：保存编辑的 Stateflow 模型。
- sfprint：打印绘制的 Stateflow 模型。

16.7 Stateflow 仿真实例

前面讲解了 Stateflow 的基本使用方法，本节将给出一个完整实际例子来演示 Stateflow 的建模与应用。这个例子是《SIMULINK-STATEFLOW TECHNICAL EXAMPLES ——Using Simulink® and StateflowTM in Automotive Applications》中的一个例子。

这个例子是关于一个弹簧——质量块系统在外力的作用下沿着平面滑动的例子，与通常的质量弹簧系统不同的是，此系统中加入了表面摩擦力。考虑表面摩擦力之后，系统就变得非常复杂。质量块与表面之间的摩擦力有阻止运动的趋势。而且摩擦力随着方向的不断变化而变化，当质量块处于静止时，摩擦力会随着外力的增大而不断增大。这就导致为

了达到系统的力学平衡，质量块在静止和滑动状态之间不断变化。

在此例中，我们用 Stateflow 来代替系统中的部分状态。正如上面的介绍，质量块和滑动面之间的摩擦力是随着它们之间的相对速度方向变化而变化的。这就使得速度和位置的连续轨迹会受到不断变化加速度的影响，我们可以将这个状态离散为“卡住”和“滑动”，我们可以通过 Simulink 中的 Stateflow 这个非常强大的工具来对离散的物理状态进行建模。

实例物理系统如图 16.45 所示。

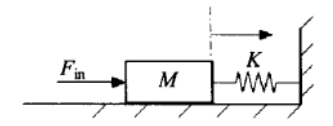


图 16.45　质量－弹簧－摩擦系统

质量弹簧基本动力学方程为：$M\ddot{x}=F_{in}-F_{sp}-F_{fr}$，其中：$M$ 为质量块质量，$x,\ddot{x}$ 分别为质量块位置和加速度，F_{in} 为外力；

F_{sp} 为弹簧力，$F_{sp}=Kx$，K 为弹簧刚度；

摩擦力 F_{fr} 比较复杂：$F_f r=\begin{cases}\text{sgn}(\dot{x})\mu F_n,|F_{st}|>\mu F\\ F_{st},\text{其他或静止}\end{cases}$，$\dot{x}$ 为速度，μ 为摩擦系数，F_{st} 瞬时摩擦力。

在许多应用当中，摩擦力被描述为静止摩擦力和运动摩擦力。在此系统中我们就假设最大静止摩擦力和运动摩擦力两个力不相同，通常最大静止摩擦力大于运动摩擦力。

$\mu F=\begin{cases}\mu_{st}F_n=F_{st},\dot{x}=0\\ \mu_{ki}F_n=F_{sl},\dot{x}\neq 0\end{cases}$，$F_{sl}$ 为滑动摩擦力。

采用下面的逻辑来得到 F_{st}。当速度为非零时，需要一个相应的脉冲力使得它瞬时为零。但是这个脉冲力通常会超过最大摩擦力，就用 F_{sl} 作为最大脉冲力。当速度保持为零时，F_{st} 就是摩擦力，使得外力合力为零，也就是使得加速度为零。

$$F_{st}=F_{in}-F_{sp}=F_{sum}$$

摩擦力就可以表示为：$F_{fr}=\begin{cases}\text{sgn}(\dot{x})F_{sl},\dot{x}\neq 0\\ F_{sum},\dot{x}=0,|F_{sum}|<F_{st}\\ \text{sgn}(F_{sum})F_{st},\dot{x}=0,|F_{sum}|>F_{st}\end{cases}$

1. 建立模型

图 16.46 就是这个系统的 Simulink 模型(model16to02.mdl)。包括两个主要部分：Mechanical Motion 模块和 state_logic 模块。前者就是普通的分级 Simulink 子系统，后者就是在 Stateflow 中执行的模块。利用 Simulink 来求解线性或者非线性常微分方程是非常理想的工具。Stateflow 演示了变换运算模式的强大功能。

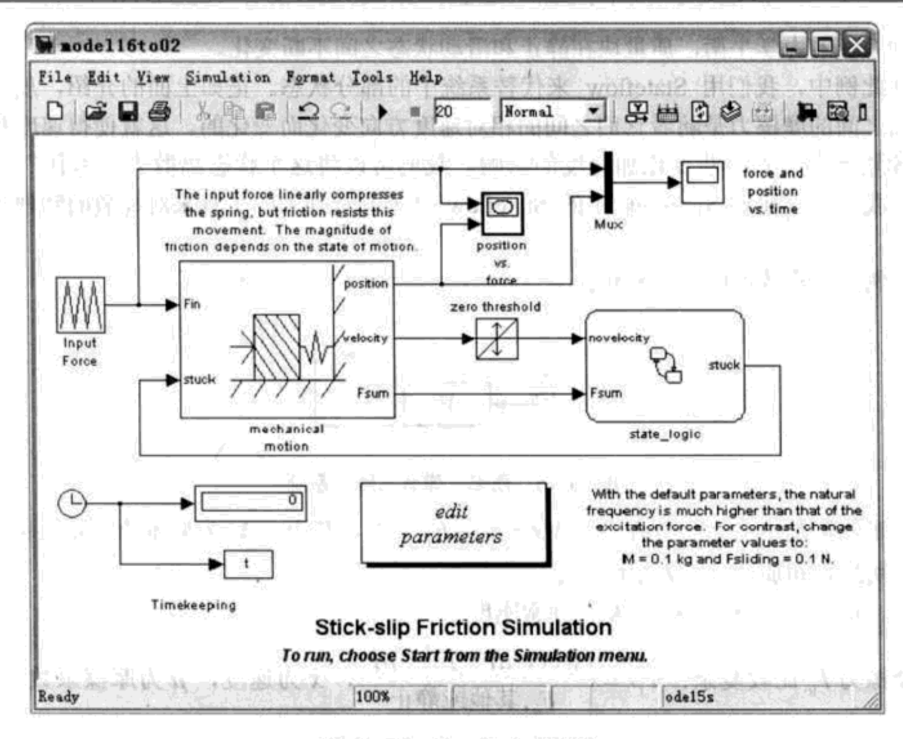


图 16.46 Simulink 模型图

双击 Mechanical Motion 模块，弹出如图 16.47 所示子系统。合力除以质量就是加速度，加速度积一次分就得到速度，再积一次分就得到位移。

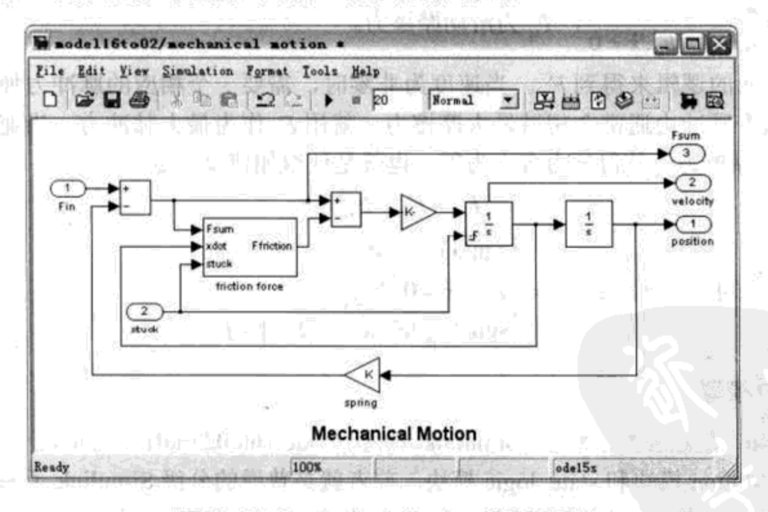


图 16.47 Mechanical Motion 模块子系统

Mechanical Motion 模块子系统中的摩擦力子系统如图 16.48 所示，里面有一系列的非线性运算，为了能够得到前面数学表达式所描述的模型。标准的 Simulink 模块有绝对值、符号、最小值以及乘。切换(switch)模块用来确定适当的摩擦力，切换(switch)模块的

控制信号是标签为"stuck"的模块。标签为"stuck"的模块是 Stateflow 控制逻辑模块的输出。也是 Mechanical Motion 子系统中第一个积分的复位输入信号。这就确保了重新从零速度开始，任何无穷小的速度值都是清零。

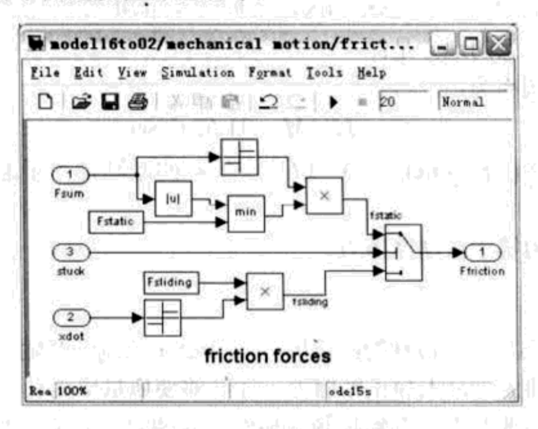


图 16.48 摩擦力子系统

如图 16.49 所示 Stateflow 图表描述了系统的状态行为。模块的输入信号是 Fsum 和 novelocity。其中 Fsum 在图 16.48 中定义了，而 novelocity 是一个二进制信号，当速度过零时就为 1。Stateflow 模块的输出就是控制信号 stuck。F_{st} 则从 MATLAB 工作控制获取。

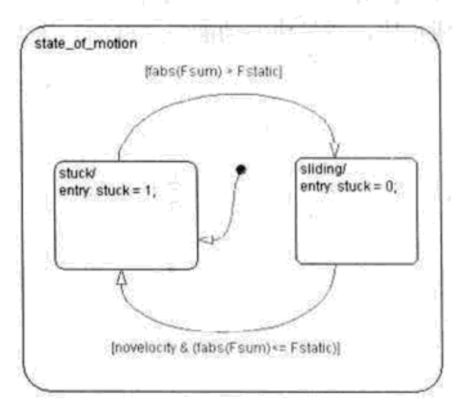


图 16.49 系统中的 Stateflow 图表

用两个异或(OR)状态来表示卡住和滑动。假定系统初始状态为静止，最开始通过空白(default)迁移将信息传递进入 Stateflow。当外力大于最大静止摩擦力时，激活出 stuck 状态起头的迁移(图 16.49 中上边的那个迁移)。只要质量块速度不为零，sliding 状态就处于激活状态。当外力小于最大静止摩擦力，即当速度为零，运动的方向有反向的趋势，此时从 sliding 状态到 stuck 状态的迁移就会执行。

stuck 的输出信号是一个代表状态的二进制数，stuck 状态的函数设置为 1，而 sliding 状态的函数设置为 0，这就可以用作 Simulink 中模块的控制信号。本例中，在满足物理定律条件下，对系统作用适当的摩擦力，状态机的最后输出就是对系统的状态迁移进行

建模。

2. 模型中默认的参数

输入力为 0N～5N，然后又从 5N～0N，周期为 5s 的折线信号。

参数有两个值得注意的地方是：

- 系统的自然频率为：

$$w_n = \sqrt{K/M} = 31.6\text{rad}/\text{sec}$$

远大于激励的频率($2\pi/10\text{rad}/\text{sec}$)。取这个低频的激励是为了能够观察到系统的静止特性。

- 最大静摩擦力等于滑动摩擦力。

3. 仿真结果

图 16.50 和图 16.51 就是仿真的结果图。图 16.50 绘制的是输入激励的曲线和位移的时程曲线。输入力必须大于最大静止摩擦力，才能够使质量块运动。在 $1 < t < 5$ 期间，位移随着弹簧力一起不断地增大，直到弹簧力和外部激励的合力小于摩擦力为止，位移不断增大的过程中，伴随着小小的振荡，这是质量块速度以系统固有频率变化的结果。输入力开始降低，直到 $t=7$ 时，质量块又开始移动，不过是向反方向移动。

类似的运动特性会再次随着作用力的降低到零又开始另外一个新的循环，图 16.51 就显示了位移随作用力的变化曲线。从图中可以看出有一个时滞循环，表示有记忆特性。

位置和速度这两个连续状态在某种意义上就具有记忆特性，具有存储能量的作用。弹簧的势能与位移的二次方成正比，质量块的动能与速度的二次方成正比。

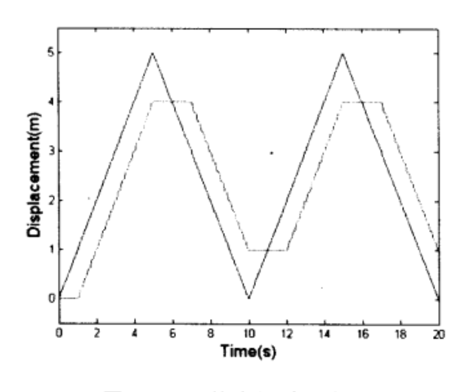


图 16.50 仿真的时程曲线

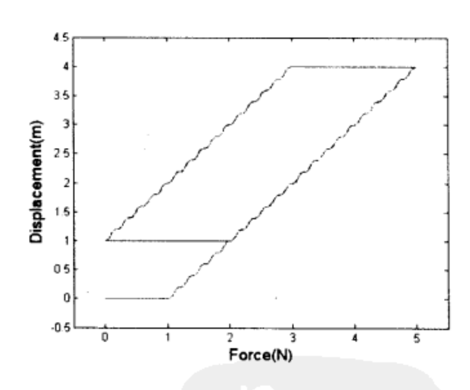


图 16.51 位移随作用力的变化

静态行为不仅依靠弹簧的刚度，而且还受到位移和速度的影响。当 stuck 状态处于激活状态时，位移保持不变。当处于 sliding 状态时，位移则依赖于弹簧的刚度、速度的方向和此时质量块所处的位置。

可以通过改变系统的参数，来观察系统状态的变化。

$M = 0.1\text{kg}$

$K = 1\text{N/m}$

$F_{st} = 0.1\text{N}$

$F_{sl} = 0.1\text{N}$

结果如图 16.52 和图 16.53 所示。

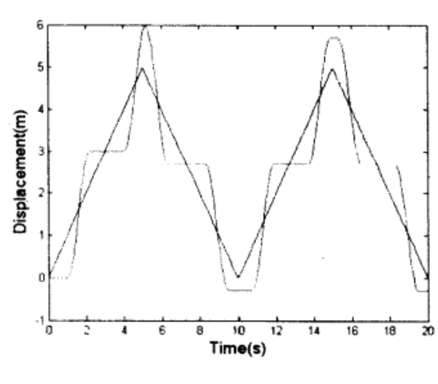


图 16.52 仿真的时程曲线

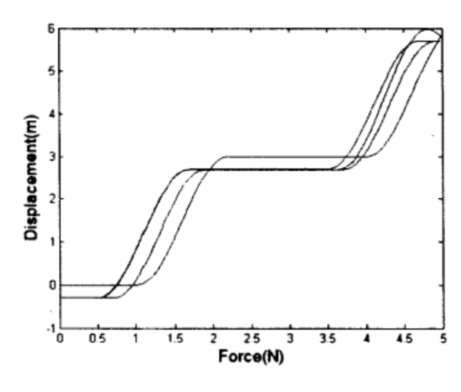


图 16.53 位移随作用力的变化

4. 结论

将 Stateflow 模块引入模型和仿真当中，可以大大地降低系统的复杂性。直观上，Stateflow 利用图形图表的直接方式来表示复杂的动力学行为，易于理解和阅读。我们可以直接将 Stateflow 插入到 Simulink 图中，实现代码生成，编译和连接的无缝融合，提供了一个非常强大的仿真环境。

第 17 章　SimMechanics 机构系统应用

通过本章的介绍，首先了解 SimMechanics 工具箱主要针对什么系统进行仿真，然后说明 SimMechanics 所具有的工具箱。在此基础上对一个简单的单摆系统进行仿真，说明仿真的基本过程。然后讲述如何通过可视化的方法查看仿真结果。为拓宽视野，加深了实例的复杂性，对一个四连杆机构进行仿真。至此，可根据所讲解实例的基本步骤和基本方法来对一个更为复杂的机构系统进行仿真，做到举一反三。

本章主要内容包括：

- SimMechanics 介绍
- SimMechanics 模块介绍
- 建立一个简单的机构实例
- 单摆运动可视化
- 四连杆结构仿真实例

17.1　关于 SimMechanics

在使用 SimMechanics 之前，我们必须明白 SimMechanics 的概念以及功能？下面就分成两节来讲述：

- 在“SimMechanics 的概念”一节中介绍 SimMechanics 和物理建模环境。
- 在“SimMechanics 的功能”一节中讲述 SimMechanics 所能够完成的内容，包括一些特殊的仿真特性。

17.1.1　SimMechanics 的概念

SimMechanics 结合 Simulink 和 MATLAB，对一个机械系统进行建模仿真。利用牛顿定律，SimMechanics 利用模块框图的建模环境来对刚体机构运动进行设计和仿真。在 SimMechanics 环境中，对机构系统进行建模和仿真，有一套工具来设定构件的各种特性，如质量特性，可能的运动，运动约束，坐标系统，还可以初始化和测量机构系统的运动。通过一系列相关联的模块来表示一个机构系统，就如同 Simulink 模型一样，而且你可以将 SimMechanics 作为 Simulink 的一个分级子系统嵌入 Simulink 当中。在仿真之前，利用 MATLAB 图形系统，SimMechanics 可视化工具可以简化为机械结构的直观显示。

SimMechanic 是物理建模的一部分，利用物理原理来对系统进行建模和设计。实际建模是在 Simulink 环境当中，可以和 Simulink 中的其他工具箱实现无缝结合。不像 Simulink 中的其他模块，SimMechanic 是直接通过模块对实际构件和构件之间的关系进行建模。

17.1.2　SimMechanics 的功能

SimMechanics 是一个模块库，库中模块使用环境为 Simulink，都具有特殊的仿真性质。通过特殊的 Sensor 和 Actuator 模块来连接 Simulink 中的标准模块。

SimMechanics 模块库中的模块都是建立机构系统所必须的。无论机构中有多少个刚体，联接和自由度，SimMechanics 都能够通过库中的模块有组织的联接来表示实际系统。

利用 SimMechanics 来对机械系统进行建模。SimMechanics 通过模块库拓宽了 Simulink 的功能，可以通过机械系统的构件以及相应的特性来求解运动方程。

利用 SimMechanics 建造机器模型操作步骤如下：

(1) 确定构件的固有属性，自由度数，约束以及坐标系统。

(2) 装配传感器和作动器，用来记录和初始化刚体运动，以及施加力或力矩。

(3) 开始仿真，调用 Simulink 的求解器来计算系统的运动，并保持约束不变。

(4) 在建模和仿真过程中，利用 SimMechanics 可视化窗口进行机械系统可视化。

17.2　SimMechanics 模块

SimMechanics 模块组提供了建模的必要模块，可以直接在 Simulink 中使用。SimMechanics 支持用户自定义的构件模块，可以设定质量和惯性量。通过节点联接各个构件来表示可能的相对运动，还可以在适当的地方添加运动约束。

打开模块组，如图 17.1 所示，包含有刚体子模块组(Bodies)、运动铰模块组(Joints)，约束与驱动模块组(Constraints&Drivers)、传感器和作动器模块组(Sensors&Actuators)、力单元模块组(Force Elements)、辅助工具模块组(Utilities)、演示模块组(Demos)。下面分别介绍。

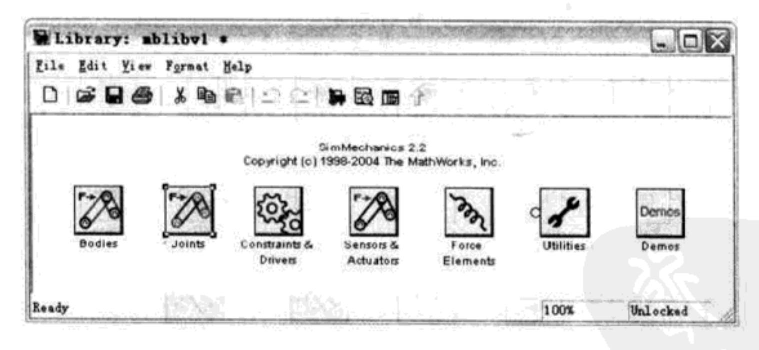


图 17.1　SimMechnics 模块组

1. 刚体子模块组(Bodies)

双击此模块，弹出如图 17.2 所示模块组。此模块组包括 3 个模块：机械环境(Machine Environment)、机架(Ground)和刚体(Body)。机械环境是为仿真定义一个环境，和它相连的有重力、维数、分析模式、约束求解器、误差、线性化和可视化。机架只有一个连接端，另外一端固定。刚体由两个连接端，其中一端为主动端，另一端为从动端。使用刚体时，

可以定义质量、惯性矩、坐标原点，还可以设定刚体的初始位置和角度。

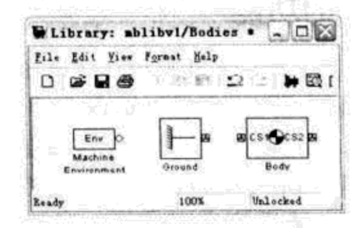


图 17.2　刚体子模块组

2. 运动铰模块组(Joints)

双击此模块，弹出如图 17.3 所示模块组。此模块组中提供了各种运动铰，利用这些运动铰就可以将刚体构件连接起来。其中包括单自由度运动铰(Prismatic)、单自由度转动铰(Revolute)、球面铰(Spherical、有三个自由度)、平面铰(Planar)、万向铰(Universal，旋转两个角度)、柱面铰(Cylindrical)、万向铰(Gimbal，旋转三个角度)、自定义铰(Custom Joint)、刚结点(Weld)。还有 Telescoping(一个方向移动，一个方向转动)、In-plane(平面内移动)、Bushing(三个方向移动，三个方向转动)、Bearing(三个方向转动)、Six-Dof(六个自由度)、Screw(螺旋铰)。

打开其中包含的两个模块组 Disassembled Joints 和 Massless Connectors。

- 双击 Disassembled Joints 模块单，弹出如图 17.4 所示模块组，其中模块是分解之后的铰，不同于图 17.3 中对应的铰，它们有不同的基准点。
- 双击 Massless Connectors 模块，弹出如图 17.5 所示模块组，其中模块是图 17.3 中对应铰的组合。

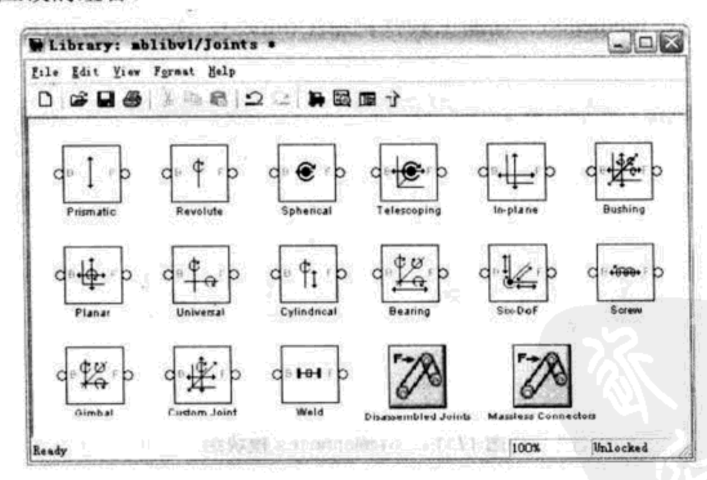
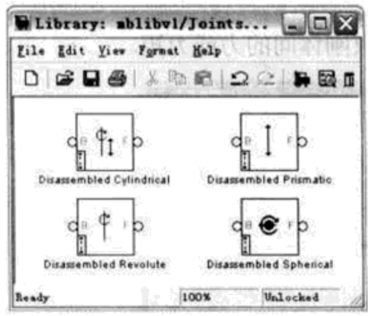
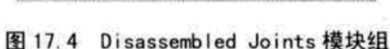


图 17.3　运动铰模块组




图 17.4　Disassembled Joints 模块组

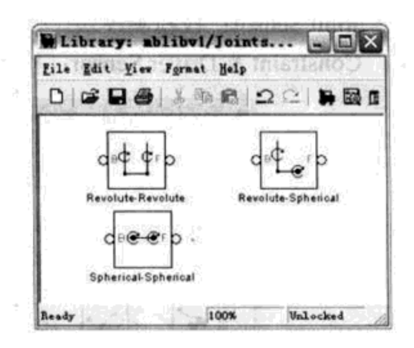


图 17.5　Massless Connectors 模块组

3. 约束与驱动模块组(Constraints & Drivers)

双击此模块，弹出如图 17.6 所示模块组。

- Distance Driver：设定两个刚体坐标原点的距离。
- Angle Driver：设定两个刚体坐标间的角度。
- Linear Driver：确定两个刚体坐标间的向量差。
- Velocity Driver：确定两个刚体坐标间的相对线速度和角速度。
- Point-Curve Constraint：曲线约束。
- Parallel Constraint：平行约束。
- Gear Constrain：齿轮约束。

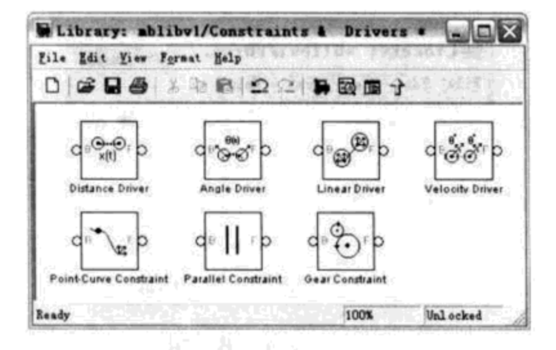


图 17.6　约束与驱动模块组

4. 传感器和作动器模块组(Sensors & Actuators)

双击此模块，弹出如图 17.7 所示模块组。该模块组中的模块用来和普通的 Simulink 模块进行数据交换。

- Body Actuator：通过广义力或力矩来驱动刚体。
- Joint Actuator：在铰接处施加力或力矩。
- Driver Actuator：对一对相互约束的刚体施加相对运动。
- Variable Mass & Inertia Actuator：在一个坐标中，刚体的质量随时间变化。
- Body Sensor：刚体检测模块。

- Joint Sensor：铰检测模块。
- Constraint & Driver Sensor：检测一对受约束刚体间的力或力矩。

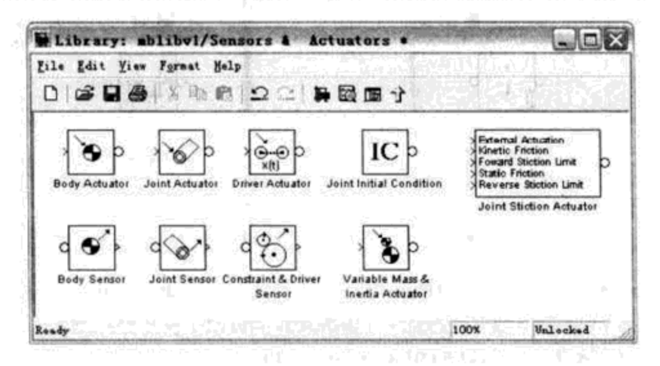


图 17.7　传感器和制动器模块组

- Joint Initial Condition：在仿真之前，给铰施加初始位移和速度。
- Joint Stiction Actuator：给铰施加摩擦力。

5. 力单元模块组(Force Elements)

双击此模块，弹出如图 17.8 所示模块组。

- Body Spring & Damper：在两个刚体之间施加线性阻尼振了。
- Joint Spring & Damper：在两个刚体间的单自由度铰或单自由度转动铰处建立一个线性阻尼振荡力或力矩。

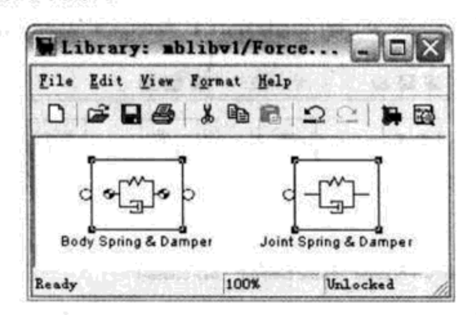


图 17.8　力单元模块组

6. 辅助工具模块组(Utilities)

双击此模块，弹出如图 17.9 所示模块组。

- Connection Port：子系统物理建模连接端口。
- Convert from Rotation Matrix to Virtual Reality Toolbox：将 3×3 的旋转矩阵转换成等价的 VRML (虚拟现实语言)的旋转轴和角的形式。
- Continuous Angle：将传感器输出的非连续、有界角度的转换成无界连续的角输出。
- Mechanical Branching Bar：将多个 sensor/actuator 映射为铰(Joint)、、约束(Constraint)、驱动器(Driver)或刚体坐标系统一个 sensor/actuator 端口。

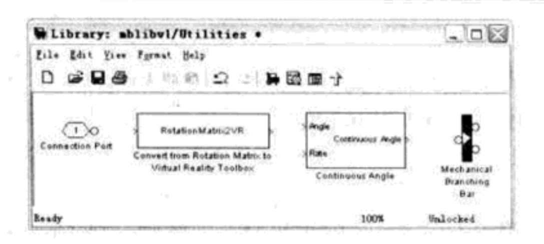


图 17.9　辅助工具模块组

17.3　建立一个简单的机构实例

有了前面介绍的 Simulink 基础，就可以比较方便地利用 SimMechanics 进行建模仿真了。

这一节主要从以下三个方面进行讲解：

- 创建 SimMechanics 模型：综述利用 SimMechanics 建立机器模型的基本步骤。
- 建立一个单摆模型：单自由度系统建模与仿真指南。
- 可视化单摆模型：利用 SimMechanics 可视化简单系统。

17.3.1　创建 SimMechanics 模型

通过这部分的介绍，将会很容易地创建一个简单模型，了解如何将 SimMechanics 模块串联起来组成一个完整的系统。

1. 建立模型的基本步骤

不管模型有多么复杂，都可以用同样的步骤建立 SimMechanics 模型。这些步骤有些类似于建造一个普通的 Simulink 模型。如果模型非常复杂，只需要添加一个步骤，而不用改变这些基本步骤。操作步骤如下：

(1) 选择 Ground,Body 和 Joint 模块：从 Bodies 和 Joints 模块组中拖放建立模型所必须的 Body 和 Joint 模块，还包括 Machine Environment 模块和至少一个 Ground 模块到 Simulink 窗口中，其中各个模块介绍如下：

- Machine Environment 模块是用来设定机器的机械参数设置。
- Ground 模块表示将一个系统固结在一个惯性坐标系中。
- Body 模块就表示刚体构件。
- Joint 模块表示互相连接构件之间的相对运动。

(2) 定位与连接模块。将 Joint 和 Body 模块拖放到适当的位置，然后按正确的顺序将它们依次连接起来，可参考如下形式：

```
Machine Env -- Ground -- Joint -- Body -- Joint -- Body -- ... -- Body
```

整个系统可以是一个开环的或者是闭环的拓扑结构，但至少有一个构件是 Gound 模块，而且有一个环境设置模块直接与其相连。

一个构件可能不只两个铰(Joint)，这就是说可以产生分支。但是铰(Joint)只能连接两

个构件。

(3) 配置 Body 模块。单击 Body 模块，打开参数对话框，配置质量属性(包括质量和惯性矩)，然后确定 Body 模块和 Ground 模块与整体坐标系或其他坐标系之间的关系。

(4) 配置 Joint 模块。单击 Joint 模块，打开参数对话框，设置移动和转动轴，以及球面铰节点。

(5) 选择、连接和配置 Constraint 模块和 Driver 模块。从 Constraints & Drivers 模块库中拖放 Constraint 和 Driver 模块到每对 Body 模块之间。打开并配置每一个 Constraint/Driver 对话框，限制或者驱动 Constraint/Driver 所连接的两个构件的相对运动。

(6) 选择、连接和配置 Actuator 和 Sensor 模块。从 Sensors & Actuators 模块库中拖放所需要的 Actuator 模块和 Sensor 模块。重新装配 Body 模块、Joint 模块和 Constraint/Driver 模块，以达到检测和驱动连接的作用。连接 Actuator 模块和 Sensor 模块，通过 Actuator 模块确定控制信号(即力、力矩或运动)，通过 Sensor 模块测量运动。

Actuator 模块和 Sensor 模块将 SimMechanics 模块和 Simulink 中的其他非 SimMechanics 模块连接起来。利用这两个模块就能够达到与 Simulink 环境实现信号传递。Actuator 模块从 Simulink 模块接受信号(例如 Simulink 中的 Sources 模块库)来激励运动。Sensor 模块的输出端口向 Simulink 中输出信号(例如 Simulink 中的 Sinks 模块库)。

在大多数机器的直接模型当中，可以首先设定力或力矩以及初始条件，然后开始仿真，得到运动结果。在运动学或者反动力学中可以通过对每一个自由度设定运动的方法，得到产生这些运动所需要的力或者力矩。

(7) 装入子系统。在 SimMechanics 模块建造的系统完成之后，就可以装入子系统作为一个模块进行调用，就如同 Simulink 中的子系统一样使用。通过 SimMechanics 中的 Utilities 模块库中的 Connection Port 模块，可以将整个 SimMechanics 模型作为子系统与一个更大的模型连接起来。

2. 配置和运行模型的基本步骤

将模块都连接好之后，此时的模型还要进一步确定如何运行，确定 SimMechanics 和 Simulink 中的设置以及装载可视化。操作步骤如下：

(1) SimMechanics 为运行机器模型提供了 4 种分析方式，最常使用的是 Forward Dynamics 方式。但是对于一个机器更加完整的分析就需要用到 Kinematics, Inverse Dynamics 和 Trimming 方式。可以对一个模型创建多个版本，在同样的基本组合结构下，为每一个版本连接不同的 Sensors 模块和 Actuators 模块以及不同的配置。

(2) 使用 SimMechanics 强大的可视化和动画显示效果。在建造模型的同时，或者模型完成之后，但必须是在开始仿真之前，可以利用可视化效果来调试机器的几何形状。还可以在仿真的同时进行动画显示。

(3) 在 Machine Environment 对话框中设定分析方式以及其他的重要机械设置。在 Simulink Configuration Parameters 窗口中运行可视化和调整 Simulink 设置。

17.3.2　建立一个单摆模型

这一部分将讲解如何拖放和配置大多数机械模型所需要的基本模块，以及如何增加传感器来测量运动，可从以下几个方面讲解：

- 总体坐标系统和重力。
- 配置 Ground 模块。
- 配置刚体(Body)模块。
- 配置铰(Joint)模块。
- 添加一个传感器(Sensor)模块，开始仿真。

1. 总体坐标系统和重力

在配置 Ground 模块之前，需要理解 SimMechanics 内部定义的绝对坐标系统称之为世界坐标系统(以后直接使用 World CS 表示)，它固定在一个惯性参考系统中。有三个坐标轴，互相垂直，满足右手法则，如图 17.10 所示。默认情况是+x 指向右方，+y 指向重力的反方向，+z 指向屏幕外。

垂直方向或是上下方向是由重力的反方向决定，重力是模型的基本属性，在仿真之前可以修改，但在仿真过程中是不能被改动的。

2. 配置一个 Ground 模块

World CS 是惟一的一个绝对坐标系统，通过它可以建立其他的若干坐标系统。还可以在 World CS 中定义一个静止的基本点，如图 17.11 所示，除 World CS 原点之外任何地方使用 Ground 模块，Ground 模块就表示基本点，同样也充当机器模型中的动力学构件的角色，好比机器环境中的一个稳定的刚体构件。

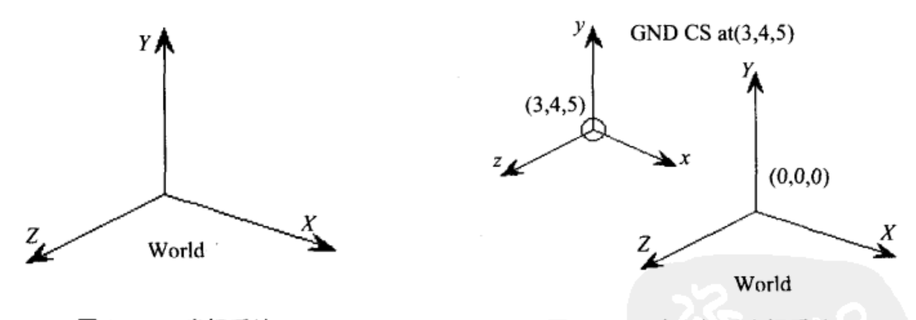


图 17.10　坐标系统　　　图 17.11　定义相对坐标系统

配置 Ground 模块的操作步骤如下：

先在 World CS 中点(3,4,5)处放置一个固定的基本点。

(1) 然后在 SimMechanics 库中打开 Bodies 库。

(2) 从 Bodies 库中拖放 Ground 模块和 Machine Environment 模块到模型窗口中，保存文件为 model17to01.mdl，然后关闭 Bodies 模块库。

(3) 打开 Ground 模块参数对话框，在 Location [x y z]栏中输入向量[3 4 5]，并选中 Show Machine Environment port 复选框，如图 17.12 所示。单击 OK 按钮，然后连接

Machine Environment 模块和 Ground 模块，如图 17.13 所示。

说明： Ground 坐标原点就是基本点，并且 Ground 坐标轴与 World CS 坐标轴平行。

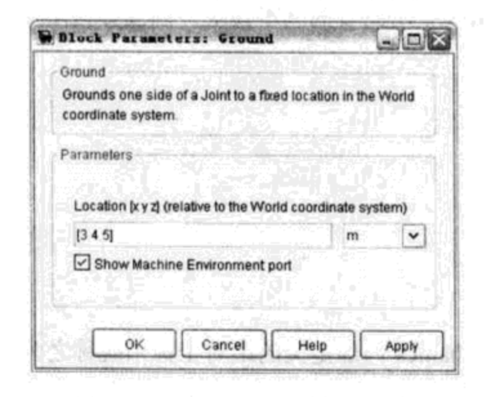


图 17.12　Ground 模块参数对话框

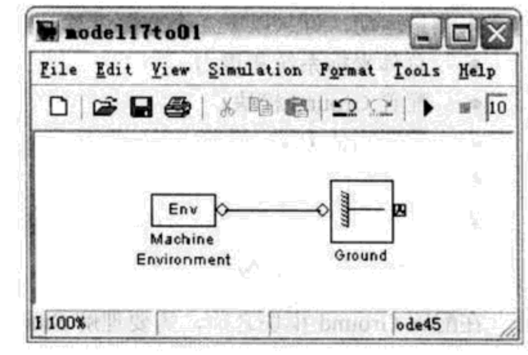


图 17.13　model17to01 模型图

3. 配置刚体(Body)模块

虽然在机器模型当中，Machine Environment 模块和 Ground 模块是必不可少的，但是一个实际系统还必须有一个或者多个刚体。所以还需要向 SimMechanics 模型中添加相应的模块来表示实际系统中的构件。这一部分就讲如何利用 Body 模块来表示实际机械系统中的刚体。

从以下几个方面进行讲解：

- Body 模块的特性。
- 单摆刚体的属性。
- Body 参数对话框的配置。

虽然刚体是机器最复杂的构件，但是 SimMechanics 不需要刚体的全部的几何尺寸和质量属性。SimMechanics 仅仅需要部分质量特性，重心，简单的几何信息，方向和随动坐标系(附着在刚体上的坐标系)。

1)　Body 模块的特性

刚体模块的主要特性包括质量特性、位置属性、方向以及随动坐标系。

质量属性包括质量和惯性张量。质量是正的实标量。惯性张量是 3×3 的对称矩阵，并不一定是对角矩阵。

从刚体重心的位置以及相对于坐标轴的方向可以知道刚体是如何旋转的。模型中刚体的初始条件，如果不加以修改，会一直保持到仿真开始。

随动坐标固定在刚体上并随着刚体一起运动，刚体最少有一个坐标，并且坐标的原点在重心。默认有三个坐标系，一个刚体重心坐标(CG CS)和两个附加坐标系为 CS1 和 CS2，分别固定在重心和刚体的两端。

除了刚体重心坐标之外，还可以添加任意的刚体坐标系统。例如，为每一个铰(Joint)、约束、驱动以及作动器和传感器配置一个坐标系统。

2)　单摆刚体的属性

这个实例中是一个简单的单摆，质量均匀，长度为 1m，直径为 2cm。初始条件为 X 轴负方向的水平位置。其中棒的一端固定在基本点(3,4,5)，坐标称之为 CS1，刚体重心坐标原点所在的重心即为几何中心，并且坐标与 World 坐标系平行。

钢棒的密度 $\rho = 7.93\text{g}/\text{cm}^3$，在刚体重心坐标中惯性张量为对角矩阵 I，其中 I_{zz} 控制关于 Z 轴的摆动，即在 X-Y 平面内的摆动。由于长度 $L = 1\text{m}$，半径 $r = 1\text{cm}$，则质量 $m = 2490\text{g}$，具体属性计算结果见表 17.1 所示。惯性张量为对角矩阵 I：

$$\begin{bmatrix} I_{xx} & & \\ & I_{yy} & \\ & & I_{zz} \end{bmatrix} = \begin{bmatrix} mr^2/2 & 0 & 0 \\ 0 & mL^2/12 & 0 \\ 0 & 0 & mL^2/12 \end{bmatrix} = \begin{bmatrix} 1250 & 0 & 0 \\ 0 & 2.08\times10^6 & 0 \\ 0 & 0 & 2.08\times10^6 \end{bmatrix}$$

表 17.1　刚体属性数据

属　性	值
质量	2490
惯性张量	$\begin{bmatrix} 1250 & 0 & 0 \\ 0 & 2.08\times10^6 & 0 \\ 0 & 0 & 2.08\times10^6 \end{bmatrix}$
重心位置	[2.5 4 5]
CS1 原点	[3 4 5]

3)　Body 参数对话框的配置

刚体 Body 与 CG 坐标，CS1 坐标和 World 坐标的关系如图 17.14 所示。

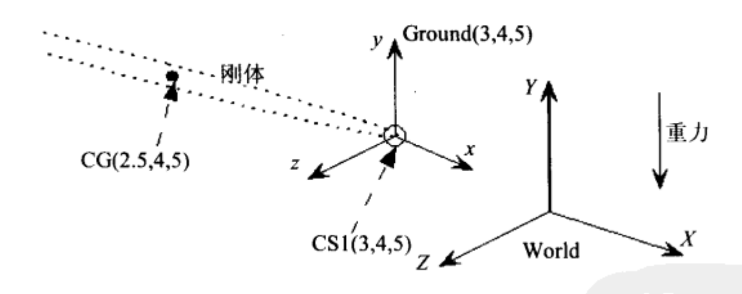


图 17.14　刚体 Body 与 CG 坐标，CS1 坐标和 World 坐标的关系图

配置 Body 模块参数的操作：

(1)　打开 SimMechanics 库中的 Bodies 模块库。

(2)　拖放 Body 模块到模型窗口。

(3)　打开 Body 模块参数对话框，注意有两栏是我们需要配置的，如图 17.15 所示。

- 质量属性——质量和惯性张量。

配置 Mass properties 选项组，其中：Mass 为 2490，单位为 g；Inertia 为[1.25e-4 0 0;0 0.208 0;0 0 0.208]，单位为 Kg*m^2。

- 刚体坐标系统——定义坐标的位置和方向。

配置 Body Coordinate Systems (Position)标签列表框，按顺序为：第一个 CG 只修改 Origin position vector[X Y Z]为[2.5 4 5]，其他设置默认；第二个 CS1 只修改 Origin position vector[X Y Z]为[3 4 5]，其他设置默认；第三个 CS2 直接删除即可。

最后配置结果如图 17.15 所示。

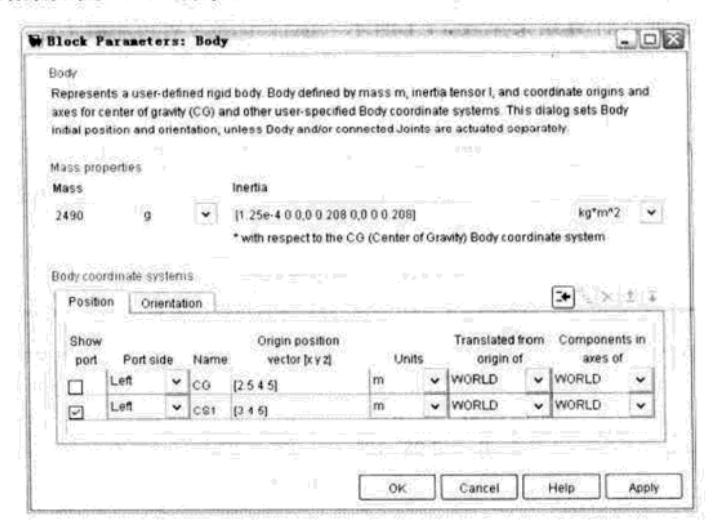


图 17.15 配置 Body Coordinate Systems (Position)面板

- 配置 Specifying Body Coordinate Systems (Orientation) 标签列表框，默认坐标平行于 World 坐标系统。

按顺序为：第一个 CG 保持默认，Orientation vector 为[0 0 0]，Units 为 deg，Relative to coordinate system 为 WORLD，Specified using convention 为 Euler X-Y-Z；第二个 CS1 保持默认，与第一个 CG 一样；最后配置结果如图 17.16 所示。

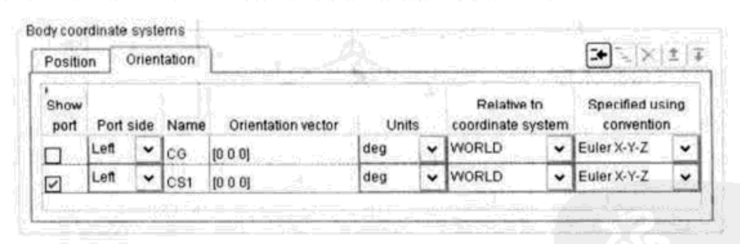


图 17.16 配置 Specifying Body Coordinate Systems (Orientation)

说明： Body 模块参数对话框中 Body coordinate system 控制面板的一排按钮的作用说明如图 17.17 所示。

4. 配置铰(Joint)模块

构成机器的刚体具有了质量和几何信息，但是刚体本身并不具备如何运动的信息。一个可能运动的方向就称之为一个自由度，这部分我们简单介绍如何利用 Joint 模块来表示

自由度。

下面分两个部分讲解：

- 如何在两个刚体间添加铰(Joint)。
- 如何为简单的单摆添加 Revolute Joint。

一对刚体中有一个可以是 Ground，不是两个同时为 Ground，另外一个刚体构件就可以定义为固定基本点进行相对运动了。固定基本点可以不是 World 坐标的原点。机器可以有多个 Ground-Body 对，但是至少要有一个。

1) 在两个刚体间添加铰

通过铰将刚体连接起来就可以表示相对运动了，一个刚体可以连接一个或多个铰。

铰模块总是连接刚体的一端，也就是刚体坐标系统的原点。那么也就是说，一个铰就是一个刚体坐标系的原点，另外一个铰就是另外一个坐标系的原点。

2) 为简单的单摆添加 Revolute Joint

尽管理解 Joint 的复杂概念比较困难，但是实际应用却是比较容易的。这一节就在前面模型的基础上，在 Body 模块和 Ground 模块之间添加一个 Joint 模块。具体关系如图 17.18 所示。

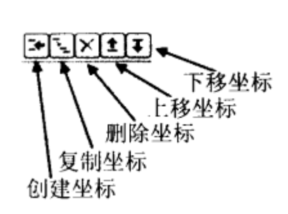


图 17.17　按钮的功能

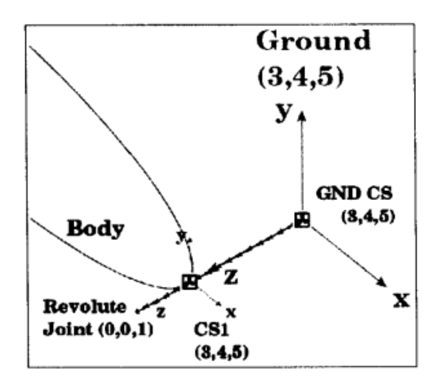


图 17.18　为简单的单摆添加 Revolute Joint

对 Revolute Joint 进行配置。铰连接的几何方式如图 17.18 所示。基本点在(3,4,5)，CS1 坐标原点在(3,4,5)，即是同一个点，但是为了清晰的理解，将它们分开了。铰连接的旋转轴是 Z 轴的正方向。

添加 Revolute Joint 模块的操作步骤如下：

(1) 打开 SimMechanics|Joints 模块库。

(2) 拖放 Revolute 模块到模型窗口。

(3) 旋转 Revolute 模块，使之可以方便地将 Joint 的 B(base)端口和 Ground(follower)模块连接起来，将 F 端口和刚体模块连接起来。这样 Ground 模块就和 Body 模块连接起来了如图 17.19 所示。

(4) 打开 Revolute 参数对话框，对 Parameters 选项组中的 Axes 选项卡进行设置，使得旋转轴为 World 坐标系统的 Z 轴，如图 17.20 所示。

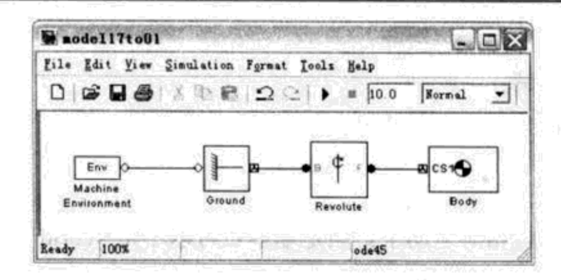


图 17.19 添加 Revolute 模块后的模型图

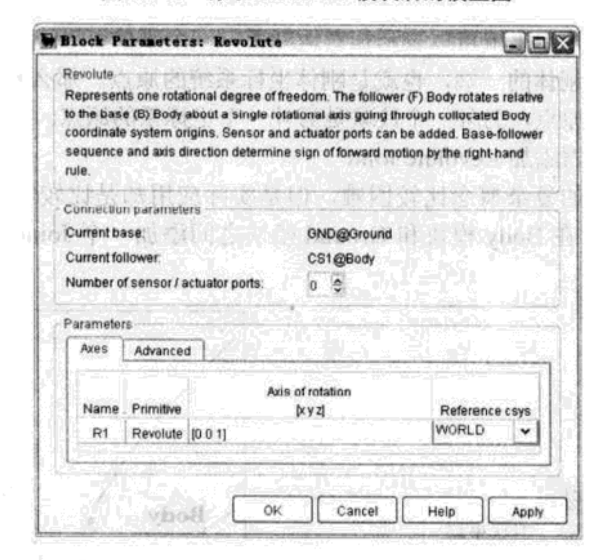


图 17.20 Revolute 模块参数对话框

应注意以下几点：

- 在 Connection parameters 选项组中，Current base 为 GND@Ground，表示用一个 Ground 模块所表示的，坐标原点在 World 系统中为(3,4,5)的接地坐标系统。
- 在 Connection parameters 选项组中，Current follower 为 CS1@Body，表示原点为 World 系统的点(3,4,5)，Body 上的随动坐标系统。
- 从 Ground 是沿着 Z 轴传递到 Body。

至此，已经完成了最简单一个机械模型，由 Ground—Joint—Body 连接组合而成。

5. 添加一个传感器(Sensor)模块并开始仿真

为了测量单摆的摆动，还必须向模型中添加 Simulink 中的 Scope 模块。Actuators & Sensor 模块库就可以提供这些模块，以便 Simulink 从 SimMechanics 模型中获取信号或者 Simulink 向 SimMechanics 输入信号。在仿真开始时，Sensor 模块可以观测到机构的运动。

下面从 3 个方面介绍具体的用法：

- 连接和配置单摆的 Sensor 模块。
- 配置 SimMechanics 中的 Machine Environment 参数和 Simulink 中的 Configuration

Parameters 参数。

- 开始仿真。

操作步骤如下：

(1) 连接和配置单摆的 Sensor 模块。此例中是测量单摆运动的角位移：

① 在 SimMechanics 库中，打开 Sensors & Actuators 模块库，拖放 Joint Sensor 模块到模型窗口中。

② 打开 Revolute 模块参数对话框，将 Number of sensor/actuator ports 原来的 0 改为 1，Revolute 模块就多了一个连接端口(空心○)。单击 OK 按钮关闭参数对话框。

③ 将这个空心端口与 Joint Sensor 模块端口相连，空心的端口就会变成实心端口●。

④ 打开 Joint Sensor 模块参数对话框，选中 Angle 和 Angular velocity 复选框，取消其他默认选项，单击 OK 按钮关闭参数对话框。

⑤ 打开 Simulink 库浏览器，从 Sinks 库中拖放一个 Scope 模块和一个 XY Graph 模块到模型窗口。从 Signal Routing 库中，拖放一个 Mux 模块，然后依次连接如图 17.21 所示。

⑥ 保存模型为 model17to01.mdl，最好在建立模型时就保存。

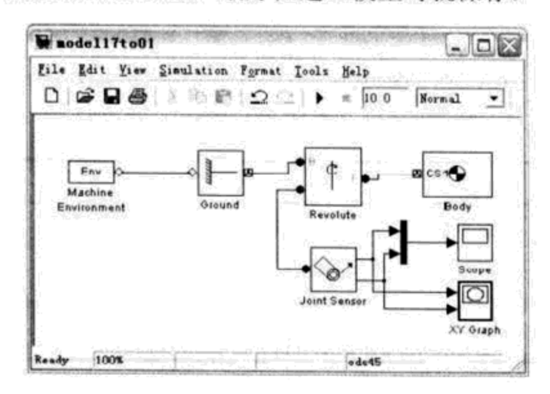


图 17.21 添加 Joint Sensor 模块后的完整模型图

(2) 配置 Machine Environment 参数和 Configuration Parameters 参数。Configuration Parameters 参数对话框对 Simulink 起着非常重要的作用，适当地设置它，可以提高仿真的精度。

① 在 Simulink 模型窗口中，选择 Simulation | Configuration Parameters 命令，打开 Configuration 参数对话框。

② 选择 Solver 选项卡，设置 Relative tolerance 为 1e-6，Absolute tolerance 为 1e-4。

③ 如果希望仿真在有限的时间内停止，可以在 Stop time 栏中设定一个有限的数值。单摆的周期约为 1.6 秒。

④ 关闭 Configuration 参数对话框。

对 Machine Environment 参数进行如下设置：

- 打开 Machine Environment 模块参数对话框，如图 17.22 所示，对话框标签的功能如表 17.2 所示。

说明： 默认的重力加速度向量为[0 -9.81 0] m/s^2，指向 y 轴的负方向。

- 关闭 Machine Environment 模块参数对话框。

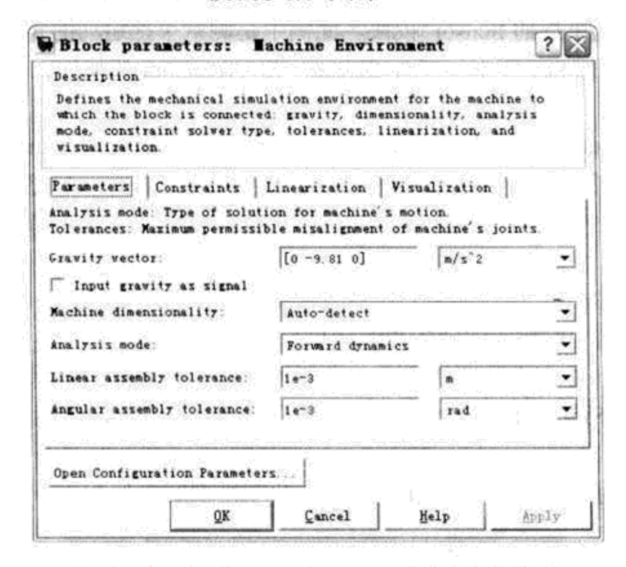


图 17.22 Machine Environment 模块参数对话框

表 17.2 标签功能

控制面板	作　用
Parameters	设定机械环境仿真的一般设置
Constraints	设定约束误差和约束求解方法
Linearization	设定如何利用 Simulink 对 SimMechanics 模型进行线性化
Visualization	设定是否对机器进行可视化

(3) 开始仿真。完成上面的建模之后，就可以进行仿真了。可以通过 Scope 模块和 XY Graph 模块观察单摆的运动。

① 打开 XY Graph 模块对话框，设定参数如表 17.3。

表 17.3 XY Graph 参数设置

参数	值
x-min	0
x-max	200
y-min	-500
y-max	500

② 打开 Scope 模块，开始仿真，XY Graph 模块会在仿真开始后自动打开。

运行结果如图 17.23 和图 17.24 所示。图 17.23 为单摆的角位移和角速度的时程曲线，图 17.24 为系统相图。

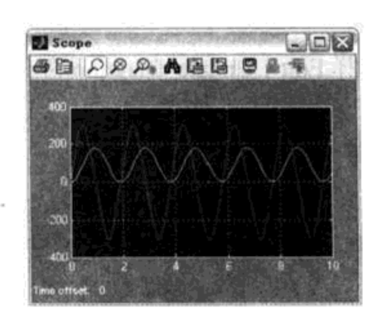


图 17.23 单摆的角位移和角速度的时程曲线

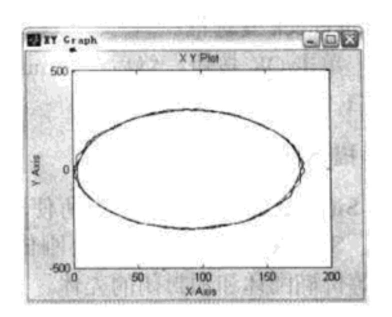


图 17.24 系统相图

17.4 单摆运动可视化

这一节中将会介绍如何利用 SimMechanics 可视化窗口将单摆运动进行可视化。在这里就利用上一节所运行的 SimMechanics 对可视化进行说明。

SimMechanics 支持自定义的 MATLAB 图像处理窗口进行可视化，这个工具以透视图的方式显示机器的运动。刚体可以通过两个方式显示，分别为等价的椭圆体或为刚体坐标中的封闭曲面。

利用两种显示方式其中的一种来解释如何可视化单摆，在仿真之前可以看到单摆，在仿真过程中可以动画显示。

1. 配置 Configuration 参数

(1) 在 Simulink 菜单中，选择 Simulation | Configuration Parameters 命令，打开 Configuration 参数对话框，选择其中的 SimMechanics 选项，如图 17.25 所示。

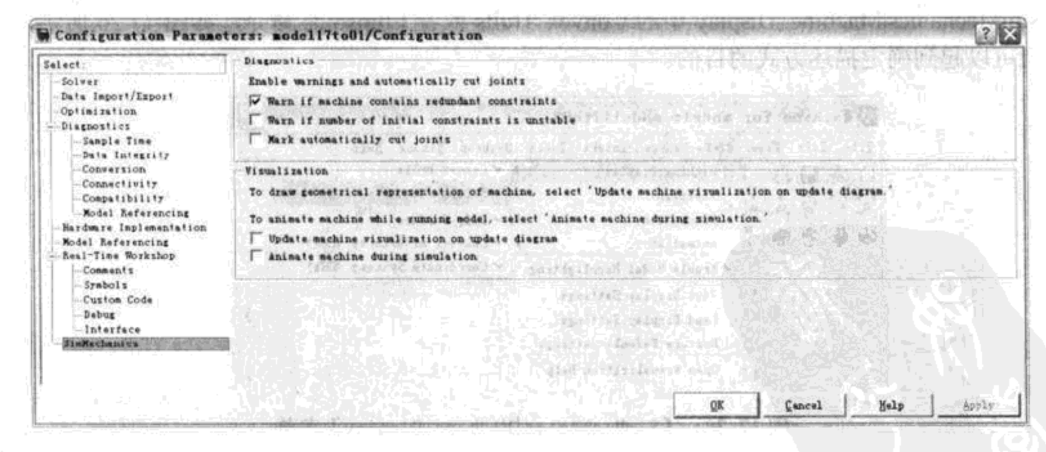


图 17.25 选择 SimMechanics 选项

(2) 为了能够观察到单摆初始静止状态，选中图 17.25 中的 Update machine visualization on update diagram 复选框。

(3) 为了能够在仿真过程中动画显示，选中图 17.25 中的 Animate machine during

simulation check 复选框。

(4) 单击 OK 按钮，然后在 Simulink 菜单中，选择 Edit | Update Diagram 命令打开可视化窗口。

2. 描述刚体

在 SimMechanics 模型中，所使用的刚体信息以某种特殊而简单的方式描述刚体已经足够了。SimMechanics 并不需要刚体全部完整的几何尺寸。下面我们分别介绍两种描述方法：等价椭圆体和和封闭的壳体。

(1) 等价椭圆体

一个刚体具有一个独一无二的等价椭圆体，这个椭圆体密度均匀，并与实际的刚体具有相同的惯性矩。

由于这个单摆钢棒是轴对称的，这里是关于 X 轴对称的，椭圆体的 3 个广义半径中的两个是相等的，即 $\partial y = \partial z$。我们可以得到广义半径为 $\partial x = \sqrt{\frac{5}{3}} \cdot \frac{L}{2} = 0.646\text{m}$，$\partial y = \partial z = \sqrt{5} \cdot \frac{r}{2} = 1.12\text{cm}$。

(2) 封闭的壳体

每一个刚体在重心(CG)处至少有一个刚体坐标系统，同时在刚体铰处还具有一个或多个额外的刚体坐标系统，还有 Actuators 模块和 Sensors 模块。通过刚体坐标系统可以定义空间中的一个体积，那么最小的外包曲面就是刚体坐标系统中的一个壳体，这个壳体也是一种可以表示刚体的方法。

在上一节中，我们创建的单摆刚体具有两个刚体坐标系统：重心坐标系统(CG)和端点坐标系统(CS1)。单摆钢棒的壳体是一个连接两个坐标原点的最小轮廓。

可以选择上面任何一种方式来描述 SimMechanics 中的刚体。在可视化窗口中选择 SimMechanics|Machine Display 中的 Convex Hulls 或者 Ellipsoids 命令，如图 17.26 所示，就可以起到确定描述方式的目的。

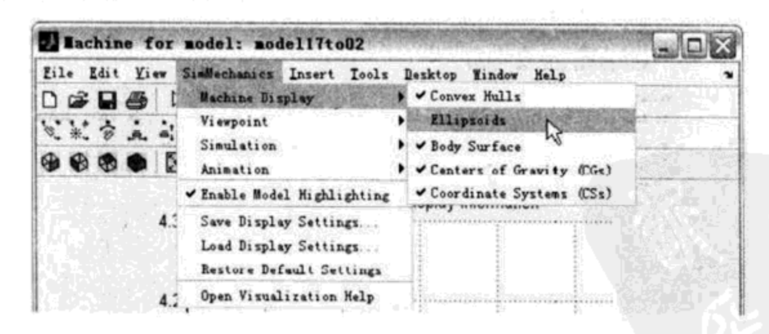


图 17.26 【SimMechanics|Machine Display】菜单

3. 利用 MATLAB 图像进行可视化

MATLAB 图像可视化工具已经嵌入到 SimMechanics，可以很方便地调用。为了能够打开这个图像窗口，或者在任何时候更新，可以在 Simulink 模型窗口中选择 Edit|Update

Diagram 命令。

- 利用壳体进行描述

在可视化窗口中选择 SimMechanics | Machine Display|Convex Hulls 命令，弹出完整的可视化界面如图 17.27 所示。

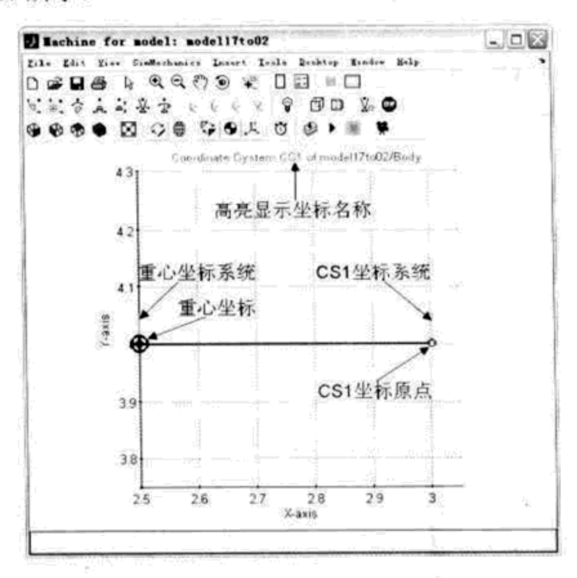


图 17.27　可视化窗口

在 Simulink 模型窗口单击运行按钮，则可视化窗口在仿真过程中与 Simulink 仿真保持同步，结果如图 17.28 所示。

- 利用椭圆体进行描述

在可视化窗口中选择 SimMechanics | Machine Display | Ellipsoids 命令，然后在 Simulink 模型窗口单击运行按钮，则可视化窗口在仿真过程中与 Simulink 仿真保持同步，结果如图 17.29 所示。

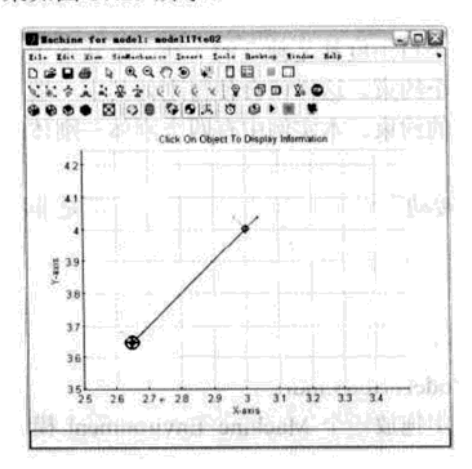


图 17.28　利用壳体进行可视化描述

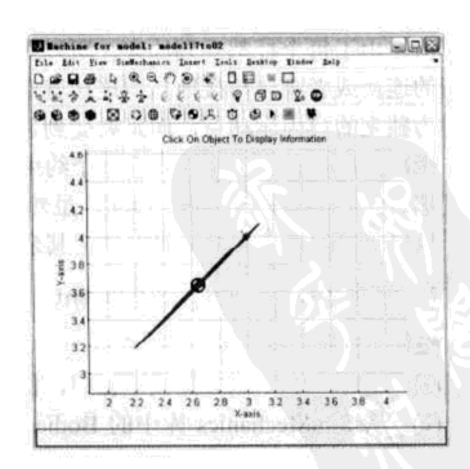


图 17.29　利用椭圆体进行可视化描述

17.5 四连杆结构仿真实例

在这个实例中将建造一个平面四连杆系统，并进一步了解 SimMechanics 的一些重要特性：

- 配置 Mechanical Environment。
- 搭建模块框图。
- 配置 Ground 模块和 Joint 模块。
- 配置 Body 模块。
- 检测运动，运行模型。

此实例有三根运动的均质钢杆，其中有两根钢杆的一端与接地点连接，第三根杆就与这两个杆剩下的端点连接起来。两个接地点就可认为是第四个杆，如图 17.30 所示，尺寸如图 17.31 所示。

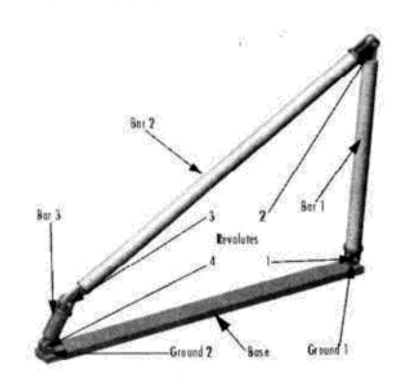


图 17.30 四连杆实物模型

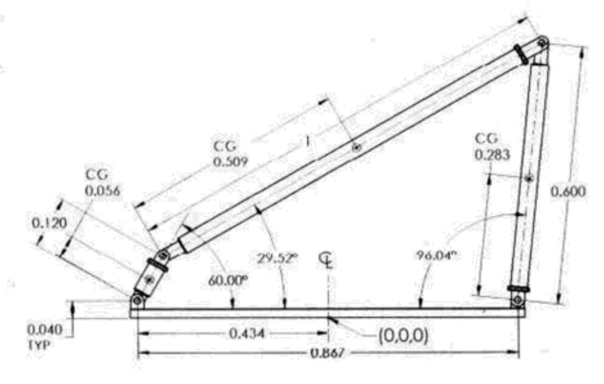


图 17.31 结构尺寸图

计算结构自由度，三个运动杆被限制到平面内运动，因此每个杆都有两个移动和一个转动，因此在考虑约束之前，自由度数是$3\times(2+1)=9$。

但是由于每个杆都受到约束，所以并不是每个自由度都是独立的。在二维状态下，刚体间的连接或者刚体与接地点的连接就会增加两个约束。这样就会使得刚体其中一端不能够作为独立的自由运动点，而是要受到邻近刚体的约束。本实例中有四个刚体－刚体或刚体－接地点的连接，这就隐含有 8 个约束。

那么最后的自由度为$9-8=1$。虽然有四个转动自由度，但是，其中三个都是非独立的，只要确定其中一个，就可以确定其余三个。

1. 配置 Mechanical Environment

操作步骤如下：

(1) 新建一个 Simulink 模型窗口，保存为 model17to03.mdl。

(2) 从 SimMechanics 库中的 Bodies 模块库中拖放一个 Machine Environment 模块和 Ground 模块到模型窗口。

(3) 打开 Ground 模块，选中 Show Machine Environment port 选项。

(4) 连接 Machine Environment 模块和 Ground 模块。

(5) 打开 Machine Environment 参数对话框，如图 17.22 所示。

(6) 在参数对话框中，设定 Angular assembly tolerance 为 1e-3 deg (degrees)。

(7) 单击 OK 按钮，关闭对话框。

(8) 在 Simulink 菜单中，选择 Simulation | Configuration Parameters 命令，打开 Configuration 参数对话框，选择其中的 SimMechanics 选项卡。

(9) 为了能够观察到单摆初始静止状态，选中 Update machine visualization on update diagram 复选框；为能够在仿真过程中动画显示，选中 Animate machine during simulation check 复选框。

(10) 单击 OK 按钮。

2. 搭建模块框图

(1) 可按照如下步骤建立模型：

机器环境→Ground 模块→铰(Joint)→刚体(Body)→铰→刚体→…→刚体→配置刚体和铰属性。

(2) 数据输入。可以直接输入数据，也可以调用 MATLAB 空间中的变量，具体见后面两小节。

(3) 搭建框图。在模块库中，代开 Bodies 库，在前面一节的基础上另外再拖放一个 Ground 模块和 3 个 Body 模块到模型窗口。然后从 Joint 库中拖放四个 Revolute 模块到模型窗口中。适当地旋转模块，并连接各个模块，如图 17.32 所示。

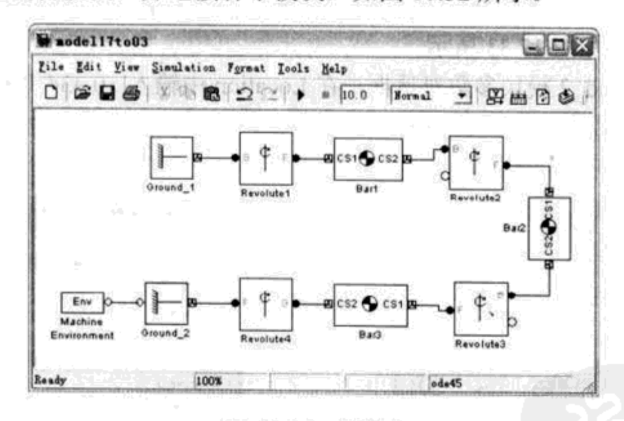


图 17.32　模型图

说明： 不同于前面的单摆模型，此模型中的四个刚体是一个闭环系统。两个 Ground 模块表示一个绝对静止的系统。而单摆系统中只有一个 Groud 模块，不能构成闭环系统。

为了保持刚体运动的一致性，要确保刚体坐标系统的一对端口必须按照 CS1-CS2, CS1-CS2…的顺序，对于每个刚杆从 Ground_1 到 Ground_2 的方向运动，在本模型中是从上到下。为了保持铰运动的一致性，铰的主(B)从(F)端口为 B-F,B-F…的顺序，同样在本模型中是从上到下。

3. 配置 Ground 模块

利用下表中的数据配置 Ground 模块，接地坐标会自动创建。

在 MATLAB 命令窗口输入以下命令：

```
>> load mat17to01
```

就会在 MATLAB 的工作空间产生所需要的数据。

表 17.4 Ground 属性值和对应的变量

属 性	值	变量名
Ground_1 point (m)	[0.434 0 0.04]	gpoint_1
Ground_2 point (m)	[-0.433 0 0.04]	gpoint_2

系统的基本尺寸为(如图 17.31)：

(1) 固定的构件长 86.7cm。

(2) Ground_1 表示接地点，在 World CS 坐标轴原点右边 43.3cm 处。

(3) Ground_2 表示接地点，在 World CS 坐标轴原点左边 43.3cm 处。

(4) 最下端的铰处于 X-Z 平面内原点以上 4cm。

为了描述稳定基础上的接地点，必须对 Ground 模块进行配置。在前面我们已经将 fourbar_data.mat 调入 MATLAB 工作空间，就可以使用这些变量了。

进行如下操作：

- 打开 Ground_1 模块参数对话框，在 Location 中输入[0.434 0 0.04]，或者变量 gpoint_1，如图 17.33 所示。
- 打开 Ground_2 模块参数对话框，在 Location 中输入[-0.434 0 0.04]，或者变量 gpoint_2，如图 17.34 所示。
- 使用默认单位 m(米)。

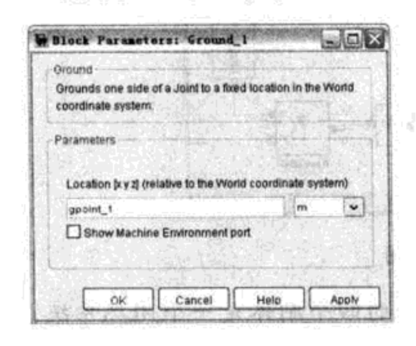


图 17.33 Ground_1 模块参数对话框

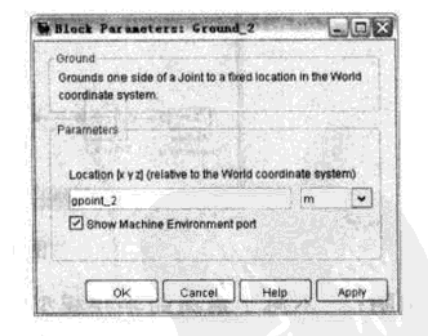


图 17.34 Ground_2 模块参数对话框

4. 配置 Joint 模块

三个没有接地的联杆都是在 X-Y 平面内的，所以 Revolute 轴必须是 Z 轴。

(1) 依次打开 Revolute 参数对话框，可以在 Axes 控制面板中看到，每个都是 Axis of rotation [x y z]默认的设置为[0 0 1]，Reference csys 都是 WORLD。

(2) 保留这些默认参数，关闭参数对话框。

依次打开 Revolute 参数对话框，在 Connection parameters 区域可以发现，Revolute 都是按照顺序排列的，分别为：

- evolute1 模块：Current Base 和 Current Base follower 分别为 GND@Ground_1 和 CS1@Bar1，如图 17.35 所示。
- Revolute2 模块：Current Base 和 Current Base follower 分别为 CS2@Bar1 和 CS1@Bar2，如图 17.36 所示。
- Revolute3 模块：Current Base 和 Current Base follower 分别为 CS2@Bar2 和 CS1@Bar3，如图 17.37 所示。
- Revolute4 模块：Current Base 和 Current Base follower 分别为 CS2@Bar3 和 GND@Ground_2，如图 17.38 所示。

Connection parameters	
Current base:	GND@Ground_1
Current follower:	CS1@Bar1
Number of sensor / actuator ports:	0

图 17.35 Revolute1 模块参数对话框 Connection parameters 参数

Connection parameters	
Current base:	CS2@Bar1
Current follower:	CS1@Bar2
Number of sensor / actuator ports:	1

图 17.36 Revolute2 模块参数对话框 Connection parameters 参数

Connection parameters	
Current base:	CS2@Bar2
Current follower:	CS1@Bar3
Number of sensor / actuator ports:	1

图 17.37 Revolute3 模块参数对话框 Connection parameters 参数

Connection parameters	
Current base:	CS2@Bar3
Current follower:	GND@Ground_2
Number of sensor / actuator ports:	0

图 17.38 Revolute4 模块参数对话框 Connection parameters 参数

5. 配置 Body 模块

对于每个 Body 模块来说都相似，但不相同，主要包括：

- 质量属性。
- 长度和方向。
- 重心位置。
- 刚体的坐标系统。

定位方式与前面单摆的例子不同，不是直接相对于 World 坐标系统，而是相对坐标表示形式，Bar1 的 CS1 相对于 Ground_1，Bar2 的 CS1 相对于 Bar1，依此类推。

表 17.5、表 17.6 和表 17.7 分别总结了每个刚体的参数。

表 17.5 Bar1 的质量和坐标数据(MKS Units)

属 性	值	变 量
质量	5.357	m_1
惯性矩	[1.07e-3 0 0; 0 0.143 0;0 0 0.143]	inertia_1
重心坐标原点	[0.03 0.282 0]相对于 CS1	cg_1
CS1 坐标原点	[0 0 0]相对于相邻的坐标系统	cs1_1
CS2 坐标原点	[0.063 0.597 0]相对于 CS1	cs2_1
重心的方向	[0 0 83.1]相对于 WORLD	orientcg_1

表 17.6 Bar2 的质量和坐标数据(MKS Units)

属 性	值	变 量
质量	9.028	m_2
惯性矩	[1.8e-3 0 0; 0 0.678 0;0 0 0.678]	inertia_2
重心坐标原点	[-0.427 -0.242 0]相对于 CS1	cg_2
CS1 坐标原点	[0 0 0]相对于相邻的坐标系统	cs1_2
CS2 坐标原点	[-0.87 0.493 0]相对于 CS1	cs2_2
重心的方向	[0 0 29.5]相对于 WORLD	orientcg_2

表 17.7 Bar3 的质量和坐标数据(MKS Units)

属 性	值	变 量
质量	0.991	m_3
惯性矩	[2.06e-4 0 0; 0 1.1e-3 0;0 0 1.1e-3]	inertia_3
重心坐标原点	[-0.027 -0.048 0]相对于 CS1	cg_3
CS1 坐标原点	[0 0 0]相对于相邻的坐标系统	cs1_3
CS2 坐标原点	[0 0 0]相对于相邻的坐标系统	cs2_3
重心的方向	[0 0 60]相对于 WORLD	orientcg_3

说明： MKS 长度为米(m)，质量为千克(kg)，惯性张量为千克。米^2 (kg-m^2)。

这 3 个刚体的配置步骤都相同，从上面 3 个表中，可以清楚地看到刚体的各种属性。配置的操作步骤如下：

(1) 设置 mass properties 区属性。将 3 个 Body 参数对话框都打开，把表中的数据对应的输入到参数对话框中，可以用数据，也可以用变量，如图 17.39、图 17.40 和图 17.41 所示。

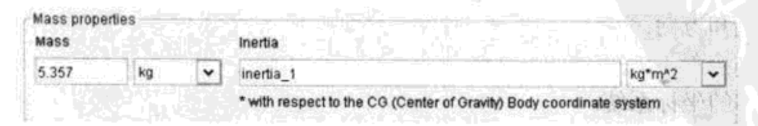


图 17.39 Bar1 的 mass properties 区属性设置

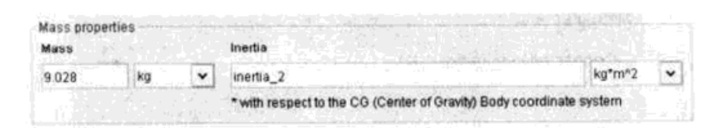


图 17.40 Bar2 的 mass properties 区属性设置

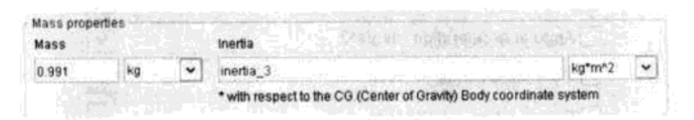


图 17.41 Bar3 的 mass properties 区属性设置

(2) 设置 Body coordinate systems 区属性。

首先设置 Position 控制面板，如图 17.42、图 17.43 和图 17.44 所示。Components in axes of 中各选项都设置为 WORLD；units 选项为默认值 m (meters)。

(3) 根据前面的表格提供的数据，设定刚体坐标属性。设定每个刚体 CG、CS1、CS2 坐标原点位置，利用表格中的数据或者变量名；设定 Translated from origin of 选项，利用表格中的信息。

6. 检测运动，运行模型

下面我们使用 Joint Sensor 模块检测铰的运动，并返回到 Simulink 中，通过 Scope 模块来显示。

(1) 添加 Joint Sensor 模块。

这里检测 Revolute2 模块和 Revolute3 模块的运动。

① 从 Sensors & Actuators 库中，拖放两个 Joint Sensor 模块到模型窗口。摆放模块，让 Joint Sensor 模块靠近 Revolute2 模块，Joint Sensor1 模块靠近 Revolute3 模块；

② 将 Joint Sensor 模块连接到 Revolute 模块之前，还必须在 Revolute 模块上增加一个端口。打开 Revolute2 参数对话框：

- 设置 spinner menu Number of sensor/actuator ports 为 1，单击 OK 按钮，此时 Revolute2 模块就会多一个端口；
- 将 Revolute2 模块和 Joint Sensor 模块连接起来。

③ 重复第二步，将 Revolute3 模块和 Joint Sensor1 模块连接起来；

④ 将 Joint Sensor 模块和 Joint Sensor1 模块的输出端口与 Scope 模块连接起来；

(2) 添加 Scope 模块

① 从 Simulink 库中的 Sinks 模块库拖放一个 Scope 模块到模型窗口；

② 双击 Scope 模块，在弹出窗口中单击按钮，在弹出对话框中设置 Number of axes 为 2，单击 OK 按钮，关闭参数对话框；

③ 将 Joint Sensor 模块和 Joint Sensor1 模块的输出端口与 Scope 模块连接起来；

④ 打开 Joint Sensor 模块和 Joint Sensor1 模块，设置如图 17.42 所示。

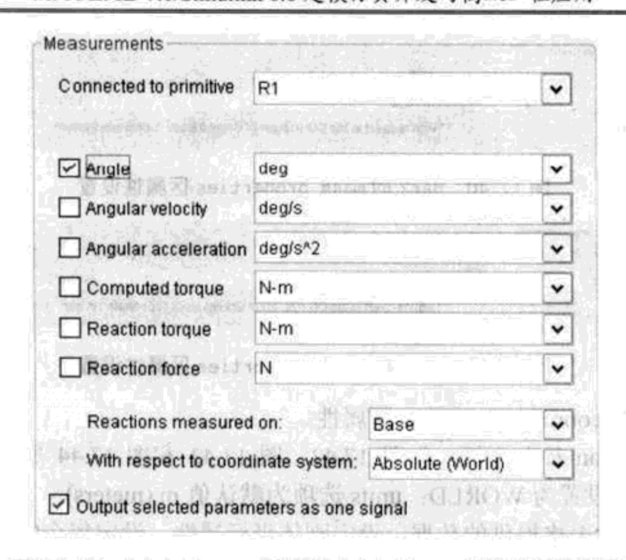


图 17.42 Joint Sensor 模块和 Joint Sensor1 模块参数设置

最后得到完整的模型图如图 17.43 所示。

(3) 配置仿真参数，运行仿真

在仿真前的最后一步就是设置 Simulink 的仿真参数，具体如下：

① 单击 Simulink 窗口菜单 Simulation|Configuration Parameters，将 Absolute tolerance 改为 1e-6，其他保持默认值，单击 OK 按钮关闭参数对话框；

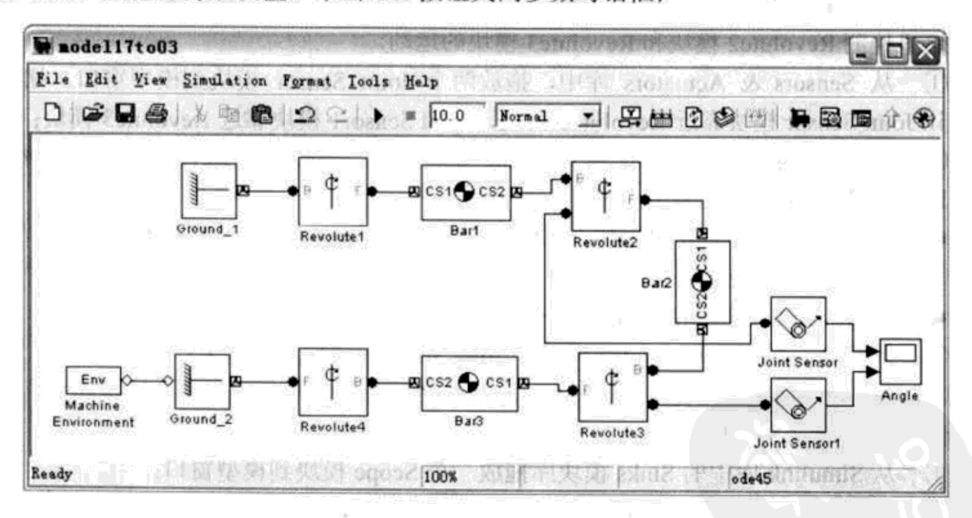


图 17.43 添加 Joint Sensor 模块后完整模型图

② 运行仿真，结果如图 17.44 和图 17.45 所示。

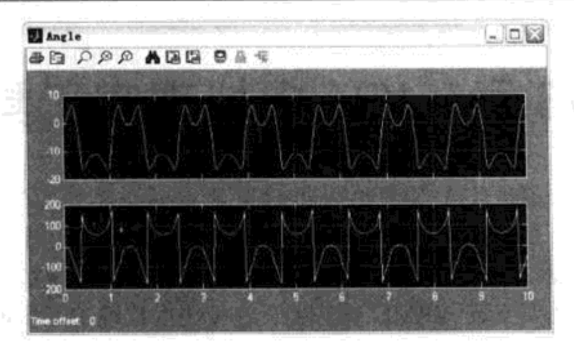


图 17.44　Revolute2 和 Revolute3 转角时程曲线

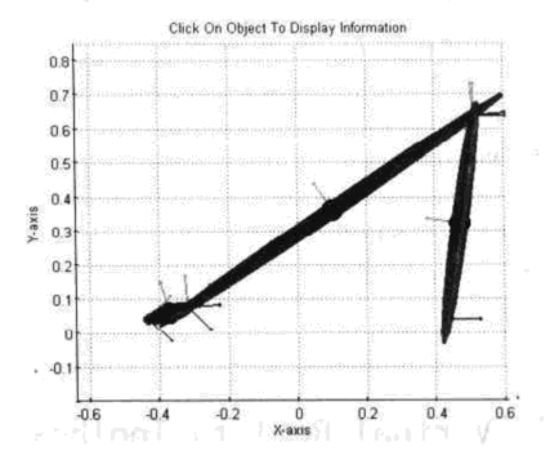


图 17.45　仿真结果动画显示



第 18 章　VRT 虚拟现实工具箱

Virtual Reality 虚拟现实工具箱可以将虚拟现实模型和动力学系统结合起来，将 MATLAB 和 Simulink 的能力拓展到虚拟现实图形中。能够营造一个使人有置身于真正的现实世界中感觉的环境。虚拟现实工具箱有别于其他工具箱，即使完全安装了 Simulink，在使用此工具箱之前还要安装 VRML 编辑器。在使用虚拟现实工具箱之前，还必须了解其与 Simulink 的接口。然后通过一个动态实例来综合了解虚拟现实工具箱的使用基本步骤和能够实现的功能。

本章主要内容包括：

- Virtual Reality Toolbox 介绍
- Virtual Reality Toolbox 功能
- 安装 Virtual Reality Toolbox
- 安装 VRML 编辑器
- VRT 虚拟现实工具箱与 Simulink 接口
- VRML 编辑工具
- VRT 虚拟现实实例
- 小结

18.1　Virtual Reality Toolbox 介绍

Virtual Reality 虚拟现实工具箱可以将虚拟现实模型和动力学系统结合起来，将 MATLAB 和 Simulink 的能力拓展到虚拟现实图形中，营造了一个使人感到置身于现实的环境，具有 5 大特征：

- 立体感的视觉效果
- 存在感
- 多感知性
- 闭环交互方式
- 动态显示

目前的研究还主要是前面 3 项，尤其是第一项。

虚拟世界是利用 Virtual Reality Modeling Language (VRML)技术创建一个虚拟和三维图形。

动力学系统是利用 MATLAB 和 Simulink 创建动力学系统。

动画是通过 Simulink 环境输出的信号可以观察到三维图形动画。

操作是在运行仿真过程中改变虚拟世界中的目标位置和属性，或者 Simulink 模型中的参数。

为了提供一个完整的工作环境，Virtual Reality Toolbox 还提供了一些额外的附件。

- VRML viewer：利用 Virtual Reality Toolbox 播放器或者在网络浏览器中加入 blaxxun Contact 插件来显示虚拟世界。
- VRML editor：对于 PC 平台，利用 V-Realm Builder 来创建和编辑 VRML 代码。对于 UNIX 或者 Linux 平台，利用 MATLAB 文本编辑器编写 VRML 代码来创建虚拟世界。

18.2 Virtual Reality Toolbox 功能

Virtual Reality Toolbox 包括了许多功能，用来创建和显示动力学系统的虚拟现实动力学模型。提供了动力学系统实时虚拟接口。

这一节主要包括以下内容：

- 支持 VRML：利用 VRML 定义虚拟世界。
- MATLAB 接口：利用 MATLAB 接口来控制虚拟世界。
- Simulink 接口：利用 Virtual Reality Toolbox 模块将 Simulink 模型和虚拟世界连接起来。
- VRML 播放器：利用 Virtual Reality Toolbox 播放器或者网络浏览器来显示虚拟世界。
- VRML 编辑器：利用 VRML 编辑工具或者文本编辑器来创建虚拟世界。
- 支持 Real-Time Workshop：支持利用 Real-Time Workshop 产生的代码进行仿真。
- 支持 SimMechanics：可以在虚拟世界中查看 SimMechanics 模型行为。
- 支持硬件：带有特殊硬件的驱动程序。
- 客户/服务器结构：为单机和服务器提供了客户/服务器结构。

1. 支持 VRML

Virtual Reality Modeling Language (VRML)满足 ISO(国际标准化组织)标准，是一种开放式的文本编辑语言。利用 VRML 定义一个虚拟世界，通过 VRML 播放器来显示，并和 Simulink 模型连接起来。

Virtual Reality Toolbox 应用了目前 VRML97 规范中的高级特性。VRML97 规范可以在网站 http://www.web3d.org 上查看。这些规范格式可以描述三维场景，声音，机内作用和万维网。

Virtual Reality Toolbox 可分析虚拟世界的结构，决定什么信号可用，然后从 MATLAB 和 Simulink 中获取这些信号。

Virtual Reality Toolbox 播放器支持大部分 VRML97 标准，几乎能够完全控制虚拟世界相关的功能。blaxxun Contact 插件支持所有 VRML97 标准。

Virtual Reality Toolbox 要确保对虚拟世界的变化必须反映到 MATLAB 和 Simulink 中。如果改变虚拟世界中的角度，则必须在 MATLAB 和 Simulink 中的虚拟目标属性得到反映。

2. MATLAB 接口

Virtual Reality Toolbox 为虚拟现实和 MATLAB 提供了一个更具有弹性的接口。创建 MATLAB 对象，并把这些对象和虚拟世界联系起来，就可以通过函数对虚拟世界进行控制。

在 MATLAB 中，可以设定 VRML 对象的位置和属性，创建 graphical user interfaces (用户图形接口 GUIs)的回调函数，并将数据映射到虚拟对象上。同样可以利用 VRML 播放器观看虚拟世界，决定其结构，并为可用的结点赋新值。

Virtual Reality Toolbox 包括找回和修改虚拟世界属性的函数，还包含保存虚拟世界结构的 VRML 文件。

MATLAB 还可以利用 MATLAB 对象来控制和操作虚拟现实对象。

3. Simulink 接口

利用 Simulink 模型，可以观察到虚拟现实的三维模型的动力学实时仿真过程。

Virtual Reality Toolbox 提供了直接用来连接 Simulink 信号和虚拟世界，这个连接以三维动画形式显示模型结果。

利用 Simulink 模块能够执行大多数 Virtual Reality Toolbox 功能。一旦在 Simulink 程序中包含这些模块，就可以将 Simulink 信号输入到虚拟世界当中。Virtual Reality Toolbox 将自动地扫描虚拟世界来寻找 Simulink 能够驱动的 VRML 结点。

所有的 VRML 结点属性都显示在分级的树状浏览器中。用户可以在 Simulink 中选择自由度来进行控制。在关闭模块参数对话框后，Simulink 就会利用在虚拟世界中所选择结点的输入输出来更新模块。在连接输入到适当 Simulink 信号，就可以在 VRML 播放器中观看仿真。

4. VRML 播放器

Virtual Reality Toolbox 自带播放器是虚拟世界的默认播放器，Virtual Reality Toolbox 播放器可以在 PC，UNIX，Mac OS X 以及 Linux 平台下使用。

如果是使用 PC 操作平台,可以在网络浏览器中安装 VRML 插件来查看虚拟世界。对于 PC 操作平台，Virtual Reality Toolbox 已经含有 VRML 插件 blaxxun 协议。

如果安装了 VRML 插件，Virtual Reality Toolbox 通过 TCP/IP 协议将 MATLAB，Simulink 与 VRML 激活的浏览器连接起来。这样不仅可以在正在运行 MATLAB 和 Simulink 的计算机上观看仿真的虚拟世界，而且可以在其他计算机上通过 Internet 来观看。

5. VRML 编辑器

对于 PC 操作平台，Virtual Reality Toolbox 包括了经典的 VRML 创建工具，由 Ligos 公司开发的 V-Realm Builder，除此之外，Virtual Reality Toolbox 提供了一个完整的 3-D 可视化仿真的创建，开发和工作环境。

利用 VRML 编辑器来创建与 Simulink 模块连接的虚拟世界。在使用之前，对于 PC 操作平台，在 Virtual Reality Toolbox 中带有 V-Realm Builder Version 2.0，也可以另选其他的 VRML 编辑器。

在编辑虚拟世界之前，必须安装编辑器，可在 MATLAB 命令窗口输入 vrinstall 命

令，具体将在后面介绍。

6. 支持 Real-Time Workshop

Virtual Reality Toolbox 已经和 Real-Time Workshop 实现了无缝结合。支持由 Real-Time Workshop 和第三方编译器产生的代码进行的仿真。同样还支持在外部目标计算机实时执行的代码。

7. 支持 SimMechanics

可以利用 Virtual Reality Toolbox 查看由 SimMechanics 创建模型的行为。首先，在 Simulink 中，通过利用 SimMechanics 模块建立机器模型。然后在虚拟世界详细确定机器的几何尺寸，将虚拟世界连接到 SimMechanics 传感器输出接口，然后在 VRML 播放器中观看动力学行为。

8. 支持硬件

Virtual Reality Toolbox 包含支持特殊硬件的驱动函数，包括 Joystick 和 SpaceMouse。利用 Simulink 模块同样可以连接到通常的硬件驱动，包括 joysticks 和 Magellan SpaceMouse。

9. 客户／服务器结构

Virtual Reality Toolbox 通过 TCP/IP 协议，将 MATLAB 和 Simulink 与 VRML 激活的浏览器连接起来。这个工具箱能够用在两种结构上：

- 单计算机：MATLAB,Simulink 和虚拟现实都在同一个主机上。
- 网络计算机：在服务器上安装 Virtual Reality Toolbox，然后通过客户端计算机上查看虚拟世界动画。可以连接多个客户端计算机。

18.3　安装 Virtual Reality Toolbox

Virtual Reality Toolbox 是随着 MATLAB6.1 版本推出的，而且越来越流行了。

18.3.1　安装工具箱

虚拟现实工具箱的安装有自己的特点，操作步骤如下：

(1) Virtual Reality Toolbox 安装。可以在 MATLAB 以前装，只要在安装 MATLAB 时选择 Virtual Reality Toolbox 就可以了。如果在安装 MATLAB 时，没有安装 Virtual Reality Toolbox，可以在重新启动 MATLAB 时安装，在安装对话框适当的地方选上 Virtual Reality Toolbox 选项，就可以安装工具箱了。

(2) VRML 浏览器安装。当安装了 Virtual Reality Toolbox，Virtual Reality Toolbox 浏览器就作为默认浏览器。如果要使用网络浏览器作为 VRML 浏览器，就必须安装 blaxxun Contact 插件，这个插件能够在 Microsoft Internet Explorer 或者 Netscape Navigator 中使用，安装步骤如下：

① 在 MATLAB 命令窗口中输入：

```
>> vrinstall -install viewer
```

就会显示：

```
Installing blaxxun Contact viewer ...
Do you want to use OpenGL or Direct3D acceleration? (o/d)
```

从上述提示中选择一个，这是选择图像加速方法，如果不确定，就选择 Direct3D，在 MATLAB 命令窗口输入：

```
d
```

就会在 MATLAB 中显示如下信息，并弹出如图 18.1 所示窗口。

```
Starting viewer installation ...
```

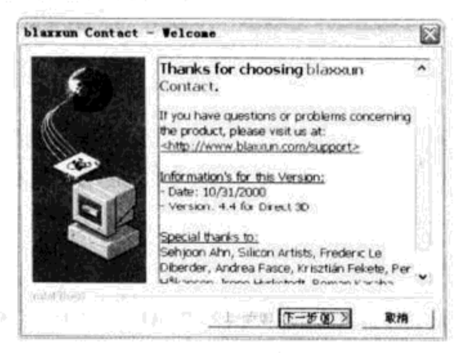


图 18.1 VRML 浏览器安装界面

② 按图 18.1 所示，按指示安装 blaxxun Contact，基本过程和其他应用软件类似，安装完成后，就会在 MATLAB 命令窗口中显示：

```
Done.
```

③ 在 MATLAB 命令窗口输入：

```
>> vrinstall -check
```

就会显示如下信息：

```
VRML viewer:   installed
VRML editor:   not installed (will be installed on first edit)
```

18.3.2 修改默认浏览器

在 IE 浏览器中使用 blaxxun Contact 插件之前，我们必须修改默认的网络安全设置，确保虚拟环境被同时更新。

更新的操作步骤如下：

(1) 打开 IE 网络浏览器。

(2) 选择菜单工具 | Internal 命令。

Internet 选项参数对话框就会弹出。

(3) 单击【安全】选项卡。

(4) 在【安全】选项卡中单击【自定义级别】按钮，就会弹出安全设置对话框。

(5) 滑动到 Microsoft VM 选项，选择如图 18.2 所示选项。在如图 18.2 窗口左下角就会出现 Java Custom Settings 按钮。

(6) 单击【Java 自定义设置】按钮，就会弹出 Local internet 对话框。

(7) 单击【编辑权限】按钮，选择未签名的内容下的启用选项，如图 18.3 所示。

(8) 在 Local internet 对话框单击【确定】按钮，在【安全设置】对话框中单击【确定】按钮，此时会弹出警告，单击【是(Y)】按钮，如图 18.4 所示。

(9) 在 internet 选项对话框中单击【确定】按钮。



图 18.2　安全设置对话框



图 18.3　编辑权限参数对话框

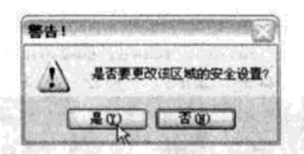


图 18.4　警告

18.3.3　设定默认浏览器虚拟环境

如果在网络浏览器中安装了 VRML 插件，就可以利用 Virtual Reality Toolbox 浏览器或者网络浏览器来观看虚拟环境。可以通过 vrsetpref 函数和 vrgetpref 函数来选择浏览器。

在 MATLAB 命令窗口中输入：

```
>> vrgetpref
```

则会出现如下信息：

```
ans =
                        DataTypeBool: 'logical'
                       DataTypeInt32: 'double'
                       DataTypeFloat: 'double'
         DefaultFigureAntialiasing: 'off'
```

```
              DefaultFigureDeleteFcn: ''
               DefaultFigureLighting: 'on'
         DefaultFigureMaxTextureSize: 'auto'
               DefaultFigureNavPanel: 'halfbar'
               DefaultFigureNavZones: 'off'
               DefaultFigurePosition: [5 92 576 380]
 DefaultFigureRecord2DCompressMethod: 'auto'
DefaultFigureRecord2DCompressQuality: 75
       DefaultFigureRecord2DFileName: '%f_anim_%n.avi'
              DefaultFigureStatusBar: 'on'
                DefaultFigureToolBar: 'on'
           DefaultFigureTransparency: 'on'
              DefaultFigureWireframe: 'off'
                       DefaultViewer: 'internal'
        DefaultWorldRecord3DFileName: '%f_anim_%n.wrl'
              DefaultWorldRecordMode: 'manual'
          DefaultWorldRecordInterval: [0 0]
              DefaultWorldRemoteView: 'off'
              DefaultWorldTimeSource: 'external'
Editor:'"%matlabroot\toolbox\vr\vrealm\program\vrbuild2.exe" "%file"'
                            HttpPort: 8123
                     TransportBuffer: 5
                    TransportTimeout: 20
                              VrPort: 8124
```

从上面信息中可以看出，DefaultViewer 属性为'internet'，表示 Virtual Reality Toolbox 浏览器为默认浏览器。就可以在浏览器中显示任何虚拟世界。

例如，在 MATLAB 命令窗口输入函数：

```
>> vrplanets
```

弹出浏览器窗口如图 18.5 所示。

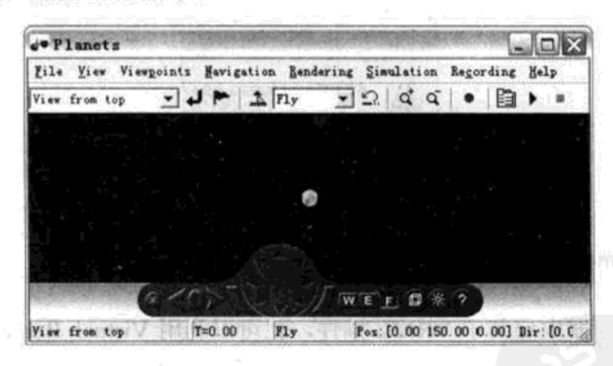


图 18.5 Virtual Reality Toolbox 浏览器窗口

修改默认浏览器，在 MATLAB 命令窗口输入：

```
>> vrsetpref('DefaultViewer','web');% 将默认的浏览器修改为网络浏览器
>> vrplanets
```

弹出浏览器窗口如图 18.6 所示，并同时会有一个 blaxxun Contact 3D console 窗口，如图 18.7 所示。

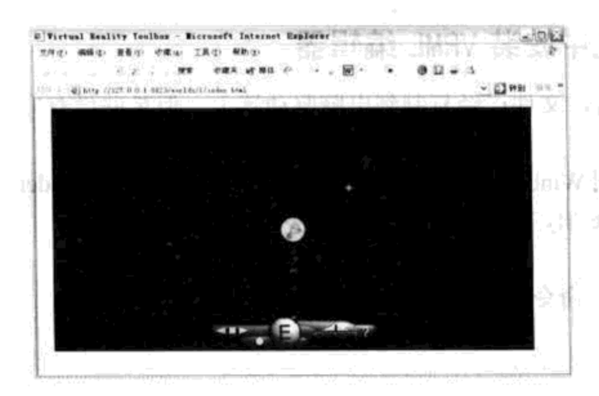


图 18.6 网络浏览器窗口

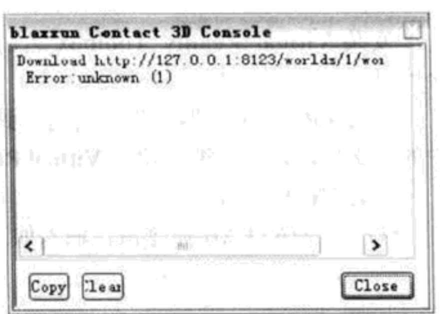


图 18.7 blaxxun Contact 3D console 窗口

如果重新设置 Virtual Reality Toolbox 为默认浏览器，在 MATLAB 命令窗口输入：

```
>> vrsetpref('DefaultViewer','factory')
```

在 vrplanets 模型的 Virtual Reality Toolbox 浏览器中，选择 Simulation| Block Parameters 命令，就会弹出 Parameters:VR Sink 参数对话框，如图 18.8 所示。

单击 View 按钮，可以看到默认设置下的目标浏览器窗口。

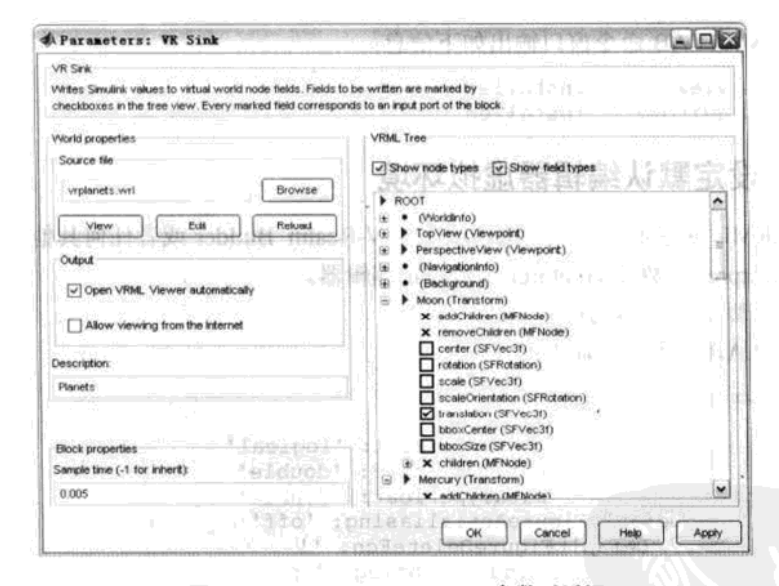


图 18.8 Parameters:VR Sink 参数对话框

18.4 安装 VRML 编辑器

可以利用 VRML 创建工具或者文本编辑器创建虚拟世界，可参见 18.4.1 节和 18.4.2 节。

18.4.1 在 Windows 操作系统中安装 VRML 编辑器

安装 Virtual Reality Toolbox 之后，文件已经复制到电脑驱动器上，但是并没有完全安装。

安装 VRML 编辑器会将密码写入到 Windows 注册表中，就可以使用 V-Realm Builder 中的库文件，在带有编辑器的 Virtual Reality Toolbox 模块中，会多出一个 Edit 按钮。

安装的操作步骤如下：

(1) 在 MATLAB 命令窗口输入如下命令：

```
>> vrinstall -install editor
```

或者

```
>> vrinstall('-install','editor')
```

就会在 MATLAB 命令窗口输出如下信息：

```
Starting editor installation ...
Done.
```

(2) 在 MATLAB 命令窗口输入如下命令：

```
>> vrinstall -check
```

就会在 MATLAB 命令窗口输出如下信息：

```
    VRML viewer:    installed
    VRML editor:    installed
```

18.4.2 设定默认编辑器虚拟环境

利用 VRML 创建工具编辑虚拟环境，如 V-Realm Builder 或者任何其他文本编辑器。可以通过 vrsetpref 函数和 vrgetpref 函数设定编辑器。

(1) 查看默认编辑器属性。

在 MATLAB 命令窗口输入：

```
>> a = vrgetpref
a =
                            DataTypeBool: 'logical'
                           DataTypeInt32: 'double'
                           DataTypeFloat: 'double'
              DefaultFigureAntialiasing: 'off'
                 DefaultFigureDeleteFcn: ''
                  DefaultFigureLighting: 'on'
            DefaultFigureMaxTextureSize: 'auto'
                  DefaultFigureNavPanel: 'halfbar'
                  DefaultFigureNavZones: 'off'
                  DefaultFigurePosition: [5 92 576 380]
     DefaultFigureRecord2DCompressMethod: 'auto'
    DefaultFigureRecord2DCompressQuality: 75
           DefaultFigureRecord2DFileName: '%f_anim_%n.avi'
                 DefaultFigureStatusBar: 'on'
                   DefaultFigureToolBar: 'on'
              DefaultFigureTransparency: 'on'
                 DefaultFigureWireframe: 'off'
```

```
                    DefaultViewer: 'internal'
        DefaultWorldRecord3DFileName: '%f_anim_%n.wrl'
              DefaultWorldRecordMode: 'manual'
          DefaultWorldRecordInterval: [0 0]
              DefaultWorldRemoteView: 'off'
              DefaultWorldTimeSource: 'external'
          Editor:
'"%matlabroot\toolbox\vr\vrealm\program\vrbuild2.exe" "%file"'
                            HttpPort: 8123
                     TransportBuffer: 5
                    TransportTimeout: 20
                              VrPort: 8124
```

(2) 查看默认编辑器。

在 MATLAB 命令窗口输入：

```
>> a.Editor
ans =
"%matlabroot\toolbox\vr\vrealm\program\vrbuild2.exe" "%file"
```

此路径是 V-Realm Builder 可执行文件的路径，V-Realm Builder 是当前 VRML 编辑器。

(3) 在 MATLAB 命令窗口输入：

```
>>vrpend
```

就会调入倒摆仿真模型，并在浏览器中显示倒摆，如图 18.9 所示。

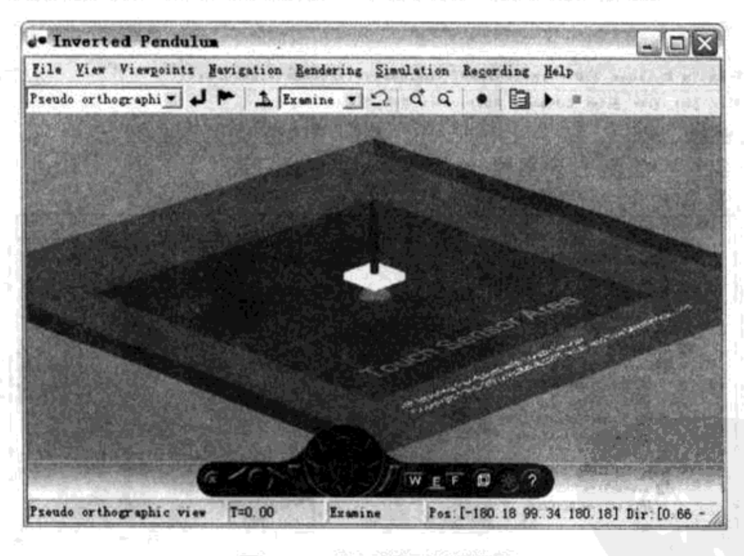


图 18.9 显示倒摆的浏览器

(4) 在 vrpend 模型的 Virtual Reality Toolbox 窗口中，选择 Simulation | Block Parameters 命令，就会弹出 Parameters:VR Sink 对话框，如图 18.10 所示。

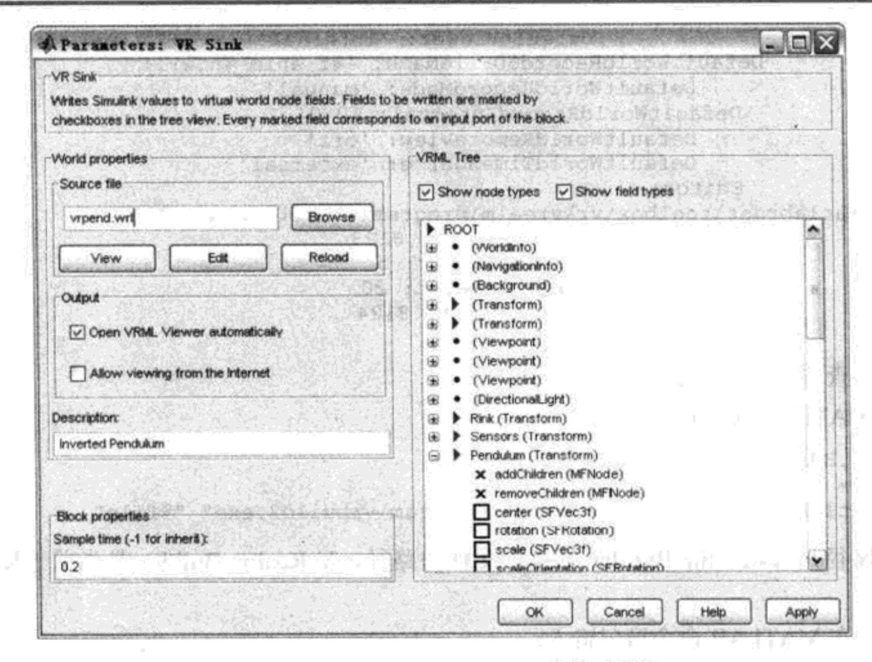


图 18.10 Parameters:VR Sink 参数对话框

(5) 单击 Edit 按钮，就会在 V-Realm Builder 创建工具中显示倒摆模型，如图 18.11 所示。

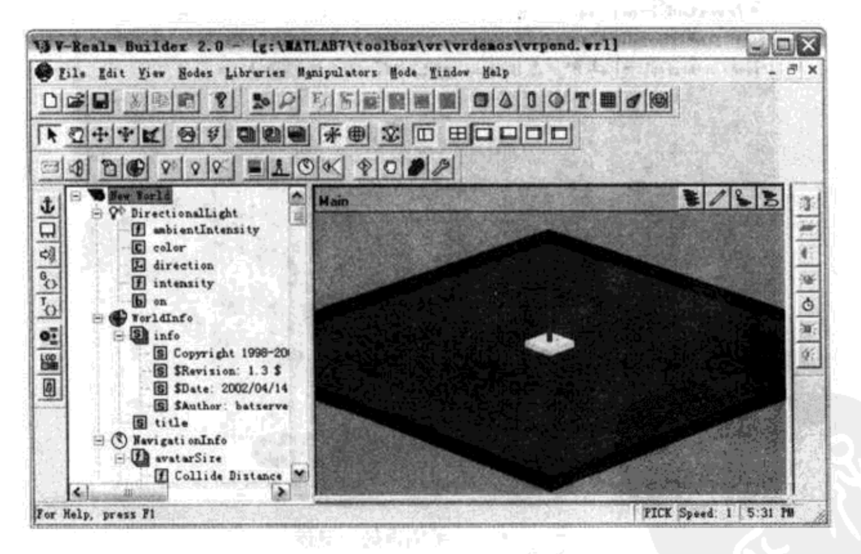


图 18.11 V-Realm Builder 创建工具

(6) 如果要修改默认编辑器为 MATLAB 编辑器，可在 MATLAB 命令窗口中输入：

```
>> vrsetpref('Editor','%matlabroot\bin\win32\meditor.exe %file')
```

(7) 重复步骤(3)~(6)，得到编辑窗口如图 18.12 所示。

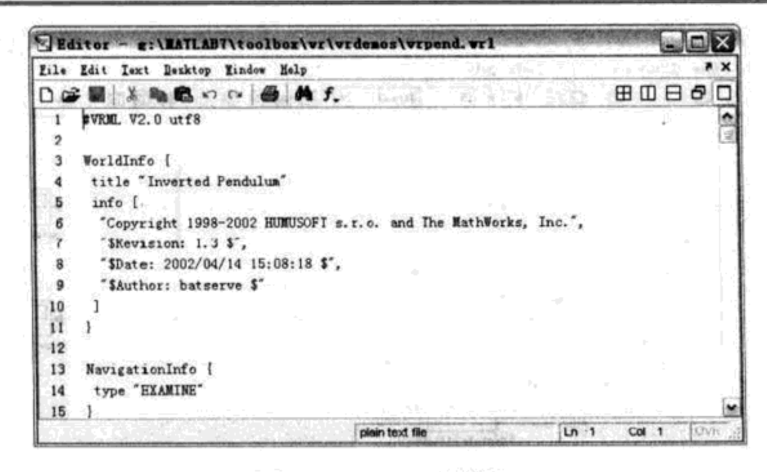


图 18.12　MATLAB 编辑窗口

18.5　VRT 虚拟现实工具箱与 Simulink 接口

Virtual Reality Toolbox 需要与 MATLAB 或者 Simulink 一起运行。然而与工具箱一起运行需要使用 Simulink 接口，这时可以直接利用图形界面接口(GUI)来使用各个工具箱的功能。

利用 Virtual Reality Toolbox，就可以将 Simulink 模块与虚拟世界连接起来。本节通过一个实例详细叙述如何在主机上仿真虚拟世界。

18.5.1　添加 Virtual Reality Toolbox 模块

将 Simulink 模型和虚拟世界联系起来，就可以利用 Simulink 模型生成动力学系统信号来控制和显示虚拟世界。创建虚拟世界和 Simulink 模型后，就可以通过 Virtual Reality Toolbox 将两者联系起来。

下面演示一个飞机起飞的过程，并可以在虚拟世界中观看。

说明： 本例中使用了 Virtual Reality Toolbox 默认浏览器。如果选择了 blaxxun Contact VRML 插件来观看虚拟世界，就必须在 Simulink 窗口开始和结束仿真，而不能通过 blaxxun Contact VRML 插件来开始和结束仿真。

(1) 在 MATLAB 命令窗口中输入：

```
>> vrtut2
```

就会打开 Simulink 模型，但是模型中没有 Virtual Reality Toolbox 模块将模型连接到虚拟世界，如图 18.13 所示。

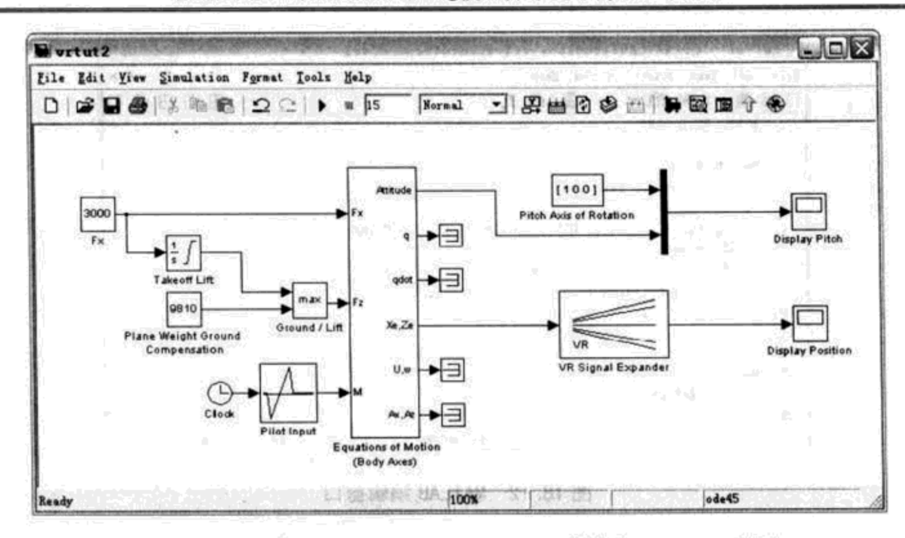


图 18.13　没有 Virtual Reality Toolbox 模块的 Simulink 模型

(2) 运行模型，通过示波器查看仿真结果。

(3) 在 MATLAB 命令窗口中输入：

```
>> vrlib
```

打开如图 18.14 所示 Virtual Reality Toolbox 模块库。

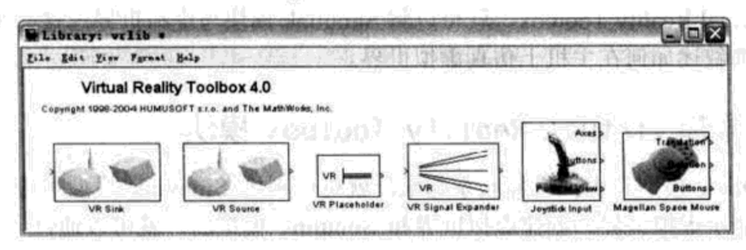


图 18.14　Virtual Reality Toolbox 模块库

(4) 从 Virtual Reality Toolbox 模块库中，拖放 VR Sink 模块到 Simulink 模型中。VR Sink 模块将 Simulink 模型数据输入到虚拟世界。关闭模块库窗口。现在已经选择了虚拟世界来可视化仿真。一个带有高速跑道和飞机的虚拟世界就会保存在 VRML 文件 vrtkoff.wrl 中，位置为 vrdemos 目录中。

(5) 双击 VR Sink 模块就会弹出 Parameters: VR Sink 对话框，如图 18.15 所示。

(6) 单击 Browse 按钮，就会弹出的 Select World 对话框中，选择<matlab root>\toolbox\vr\vrdemos 中的 vrtkoff.wrl 文件。

(7) 在 Description 文本框中，输入模型的简短描述，如 VR Plane taking off。

(8) 单击 Apply 按钮，Parameters: VR Sink 参数对话框中就会出现 VRML 树了，显示了虚拟环境的各种结构。

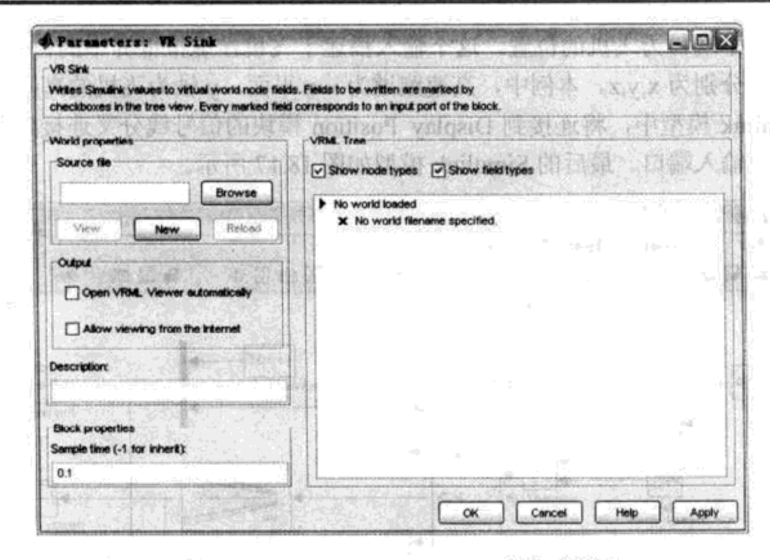


图 18.15 Parameters: VR Sink 参数对话框

(9) 打开 Plane 结点，选择 translation 和 rotation 选项。

这两个选项是显示飞机的位置和角度。如图 18.16 所示。单击 OK 按钮，VR Sink 模块就会出现两个输入端口。

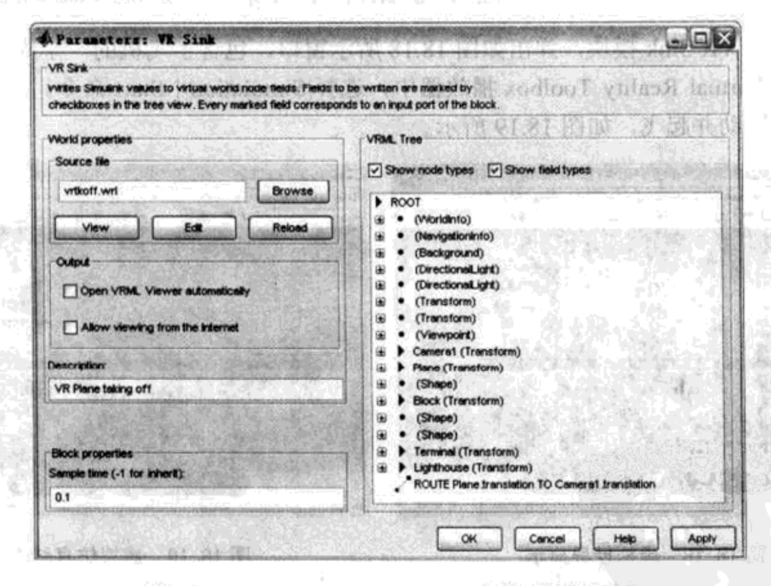


图 18.16 Parameters: VR Sink 参数对话框设置

(10) 第一个输入端口为飞机角度，旋转角度由四元素向量所定义。前面 3 个元素定义了旋转轴角度。此例中 x 轴为[1 0 0]。飞机的倾斜由 x 轴角度表示，最后一个元素是关于 x 轴的旋转角度，单位为弧度。

在 Simulink 模型中，将连接到 Display Pitch 模块的信号线分叉连接到 VR Sink 模块的飞机角度输入端口。

(11) 第二个端口为飞机的位置。这个输入描述了飞机在虚拟世界中的位置。位置由 3 个坐标表示，分别为 x,y,z。本例中，高速跑道为 x-z 平面，y 轴为飞机的高度。

在 Simulink 模型中，将连接到 Display Position 模块的信号线分叉连接到 VR Sink 模块的飞机位置输入端口。最后的 Simulink 模型如图 18.17 所示。

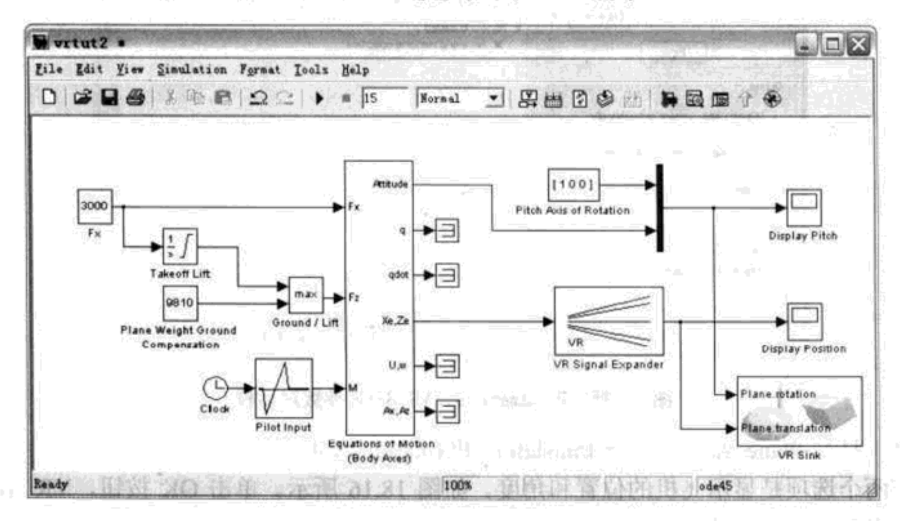


图 18.17　Simulink 模型显示

(12) 双击 VR Sink 模块，弹出如图 18.18 所示窗口，包含了飞机的一个虚拟世界。

(13) 在 Virtual Reality Toolbox 播放器中，选择 Simulation | Start 命令。开始仿真，飞机就会向左滑动并起飞，如图 18.19 所示。

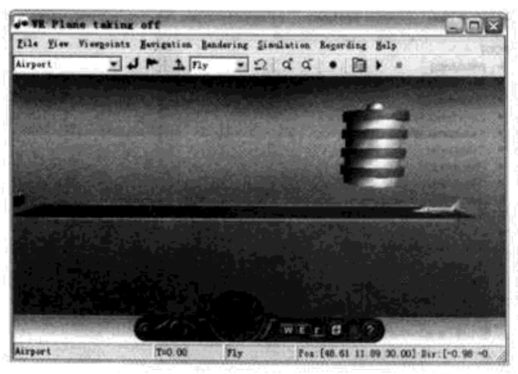



图 18.18　虚拟世界显示　　　　图 18.19　虚拟仿真结果

18.5.2　修改与 Simulink 模块连接的虚拟世界

有的时候，要将不同的虚拟世界连接到 Simulink 模型当中，或者连接不同的信号。将虚拟世界和 Simulink 模型连接好之后，还可以选择另外一个虚拟世界，并修改连接到虚拟世界的信号。本节假设将 Simulink 模型 vrtut2 连接到虚拟世界。

修改的操作步骤如下：

(1) 在上一小节的基础上，双击 VR Sink 模块，就会出现浏览器窗口。

(2) 在浏览器窗口选择 Simulation|Block Parameters 命令，就会弹出 Parameters: VR Sink 对话框。

(3) 在 Parameters: VR Sink 对话框中，单击 Browse 按钮，在弹出的窗口中选择目录 <matlab root>\toolbox\vr\vrdemos 中的文件 vrtkoff2.wrl，然后单击【打开】该文件。

(4) 单击 Apply 按钮。

(5) 在 Plane 结点下，选中 translation 复选框，清除 rotation 复选框，单击 OK 按钮。此时 VR Sink 模块就会变成只有一个模块，表示位置。

(6) 确认连接正确，应该是 VR Signal Expander 输出连接到 VR Sink 模块。

(7) 运行仿真，如图 18.20 所示。

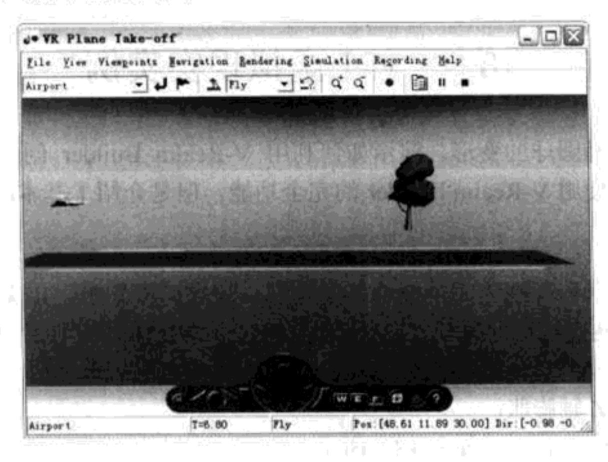


图 18.20　虚拟仿真结果

18.6　VRML 编辑工具

可以通过多种方法来利用 VRML 创建虚拟世界。例如，可以直接利用文本编辑器来编写 VRML 代码，或者用 VRML 编辑器来创建虚拟世界，而不用懂得 VRML 语言，但需要知道连接虚拟世界和 Simulink 模块和信号的 VRML 目录树的结构。

许多用户还是比较愿意选择利用自己喜爱的文本编辑器来创建简单的虚拟世界。然而最主要的方法依然是利用三维工具来创建虚拟环境。这些工具可以让你不需要知道过多的 VRML 语言就能够创建复杂的虚拟世界。

这些三维工具提供了强大而丰富的创作手法来建立实际的技术模型。例如，可以从一些 CAD 工具中容易和方便地导入三维目标模型来创建。以下介绍两个常用的三维编辑工具。

- 普通三维编辑工具

这些编辑工具本身的格式并不是 VRML。但是可以将格式扩展到 VRML 格式。有很多这些软件，例如 3D Studio，SolidWorks 或者 mantra4D 都能够进行转换。这些软件功能强大，并且使用相对容易。但又各有不同。有的侧重艺术，有的侧重动画，有的侧重游

戏，还有的侧重技术。根据应用领域不同，它们提供了不同的工作环境。有的功能强大，有的容易学习。

- 原始 VRML 编辑器

这些编辑器直接利用 VRML 作为文件格式，这就保证了编辑器中的所有特性都能够和 VRML 兼容。这些编辑器可以使用 VRML 格式所独有的特性，如分类机和传感器。

但是非常遗憾的是，目前这类高级的 VRML 编辑器还很少有商业产品。大多数此类编辑器还在起步阶段，而且比普通三维编辑工具难用。但是 Ligos 公司的 V-Realm Builder 却是一个目前可以在 PC 机上使用最高级的 VRML 编辑工具之一。目前 V-Realm Builder 只能够在 Windows 操作系统上使用。V-Realm Builder 是一个灵活性非常大的三维图像编辑工具，提供非常方便的 VRML 语法接口。

18.7 VRT 虚拟现实实例

本节介绍一个椭圆球的变形。演示如何利用 V-Realm Builder 创建一个简单的虚拟世界。本例中并没有使用 V-Realm Builder 的完全功能，但是介绍了基本的操作步骤。

1. 定义问题

假定用户要仿真和可视化一个椭圆体的变形。在虚拟世界中，定义两个块体表示刚性平面(B1, B2)，在两个平面之间定义一个球体(S)。三个物体是中心沿 X 轴对齐的。块体 B1 和块体 B2 以同样的速度向球体 S 移动。当两个块体接触到球体时，球体开始变形，沿 X 轴收缩，Y 轴和 Z 轴膨胀。

块体和球体的位置和尺寸如表 18.1 所示。

表 18.1 块体和球体的位置和尺寸

物 体	中心位置	物体尺寸
B1	[3 0 0]	[0.3 1 1]
B2	[-3 0 0]	[0.3 1 1]
S	[0 0 0]	r = 0.9

2. 添加 Virtual Reality Toolbox 模块

利用 Simulink 模型 vrtut3.mdl 来演示如何向模型中添加 Virtual Reality Toolbox 模块。这个模型产生块体 B1 和 B2 的位置和球体 S 的尺寸。

(1) 从目录<matlab root>\toolbox\vr\vrdemos\中复制文件 vrtut3.mdl 到 MATLAB 当前工作目录。

(2) 在 MATLAB 命令窗口中输入：

```
>> vrtut3
```

一个带有 Virtual Reality Toolbox 中的 VR Signal Expander 模块的 Simulink 模型窗口就会被打开，但是没有 VR Sink 模块将模型中的数据输入到 Virtual Reality Toolbox，而是使用示波器 Scope 模块来临时显示相关信息，如图 18.21 所示。

(3) 在 MATLAB 命令窗口中输入：

```
>> vrlib
```

打开 Virtual Reality Toolbox 库。

(4) 从库窗口中拖放 VR Sink 模块到 Simulink 窗口中。

(5) 将模型另存到当前工作目录，文件命为 model18to01.mdl。

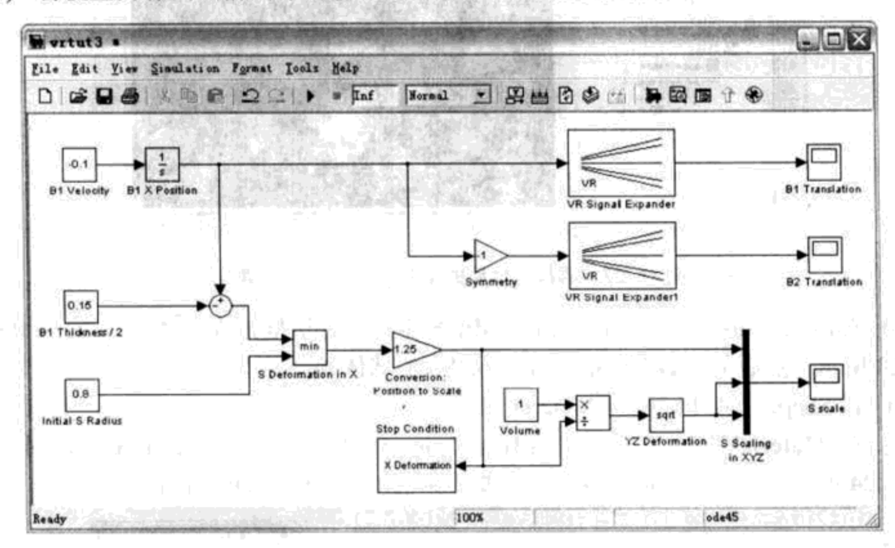


图 18.21　利用示波器 Scope 模块来临时显示相关信息

3. 在虚拟世界中创建球体

在将 VR Sink 模块添加到 Simulink 模型中后，就可以利用 V-Realm Builder 创建虚拟世界了。本部分是接着上一小节进行说明的。

(1) 选择【开始】按钮，然后单击【运行】按钮。

(2) 在运行窗口中输入<matlab root>\toolbox\vr\vrealm\program\vrbuild2.exe，单击【确定】按钮。

说明： 还可以直接在<matlab root>\toolbox\vr\vrealm\program\ 目录下单击 vrbuild2.exe 文件打开 V-Realm Builder 应用程序。

(3) 在 V-Realm Builder 应用程序窗口中，选择 File | New 命令，或者单击新建按钮。

在 V-Realm Builder 面板中，左边显示的是空的 VRML 树。

(4) 在工具栏中单击按钮，就会在 VRML 树看到球体的一些具体属性，树中包含 Transform, Shape, Appearance, Material 和 Sphere 等节点，如图 18.22 所示。

顶层节点是 Transform，这个节点包含了可以改变物体位置和比率的节点。其中一个物体构成了一个子目录树，其中 Shape 节点包含了外形和几何信息。

(5) 展开 Sphere 节点，Radius 属性就会出现，黄色的图标表明值的类型。本例中，f 表示 SFFloat 类型的值，是 32 位的浮点值。

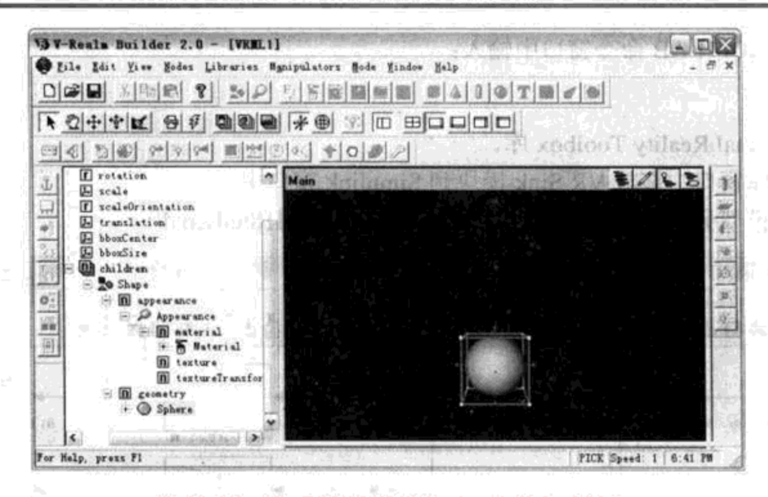


图 18.22　添加球体后的 V-Realm Builder 窗口

(6) 双击 Radius 属性，就会弹出 Edit SFFloat 对话框，在文本框中输入 0.9，如图 18.23 所示。然后单击 OK 按钮，右边窗口中的球体就会变小了。

(7) 在 Shape 节点下，展开 Material 节点。

(8) 在 Material 节点下，双击 diffuseColor 属性，就会打开 Edit Color 对话框，如图 18.24 所示。在其中可以设置任何需要的颜色，然后单击 OK 按钮。

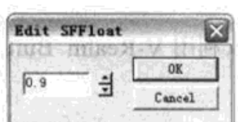
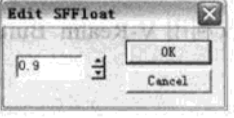



图 18.23　Edit SFFloat 参数对话框



图 18.24　Edit Color 参数对话框

(9) 如果想查看和修改球体的位置，双击 Transform 节点下面的 translation 属性。本例中无需修改默认值[0 0 0]。

本例中是要改变球体的形状，可以通过修改 Transform 节点下的 scale 属性值。Virtual Reality Toolbox 需要这个节点设定一个名称，以便能够被访问。

(10) 单击 Transform 节点，然后再单击，名称就处于可编辑状态，在这里输入 S。

4. 在虚拟世界中创建块体

(1) 单击目录树中最顶层的 New World，然后在工具栏中单击■按钮。

一个新的块体就会出现在和球体同一个位置，由于块体比较大，球体暂时不可见。

(2) 双击属于块体的 Transform 节点下的 translation 属性，就会打开 Edit Vector 3 参数对话框。

(3) 选中 X Axis 复选框，然后在编辑框中输入 3，如图 18.25 所示，然后单击 OK 按钮。

(4) 展开 Box 节点，双击其他的 size 属性，Edit Vector 3 参数对话框就会打开，设置为[0.3 1 1]，如图 18.26 所示，然后单击 OK 按钮。

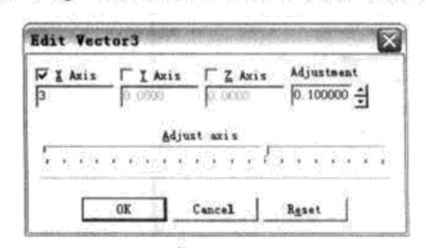


图 18.25 位置 Edit Vector 3 参数对话框

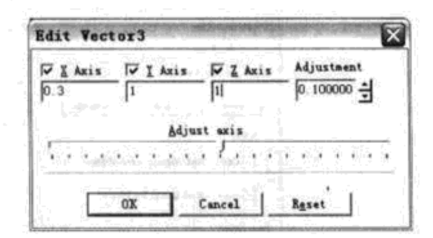


图 18.26 尺寸 Edit Vector 3 参数对话框

(5) 为了区分，还需要设置颜色，在 Material 节点下双击 diffusecolor 属性，设置相应的颜色，然后单击 OK 按钮。

(6) 单击 Transform 节点，然后再单击，名称就处于可编辑状态，在这里输入 B1。

(7) 重复步骤(1)～(6)，其中步骤(3)在编辑框中输入-3，步骤(6)在编辑框中输入 B2。结果如图 18.27 所示。

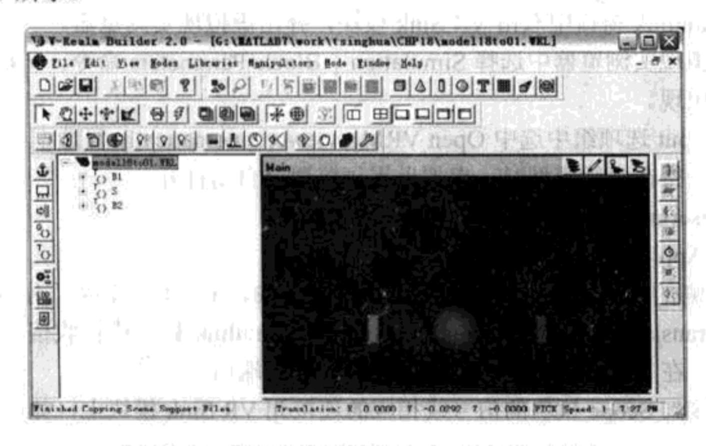


图 18.27 添加块体后的 V-Realm Builder 窗口

(8) 保存虚拟世界为 model18to01.wrl，必须和 vrtut3.mdl 在同一个目录下，然后退出 V-Realm Builder。

5. 连接 Simulink 模型与虚拟世界

在创建好虚拟世界和 Simulink 模型，并向模型中添加 Virtual Reality Toolbox 模块到模型中后，就可以在 Simulink 与虚拟世界之间定义一个连接。这一部分就以 model18to01.mdl 模型作为实例，假设现在已经打开模型，并已经向模型中添加了 VR Sink 模块。操作步骤如下：

(1) 在 Simulink 窗口中双击 VR Sink 模块，在弹出的 VR Sink 模块参数对话框中，单击 Browse 按钮，在弹出文件打开窗口，选择保存的 model18to01.wrl 文件，单击 OK 按钮，如图 18.28 所示，在右边窗口会出现相应的节点内容。

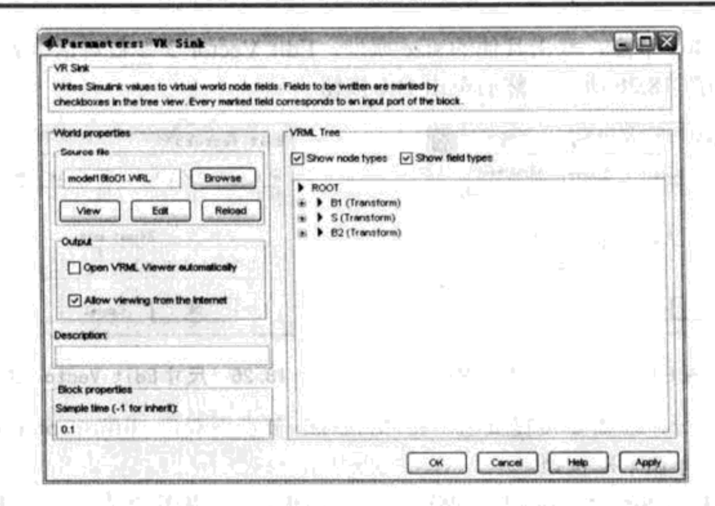


图 18.28 打开 model18to01.wrl 文件

(2) 在 VR Sink 模块参数对话框中单击 OK 按钮。

(3) 在 Simulink 窗口中双击 VR Sink 模块，弹出虚拟现实浏览器。

(4) 在虚拟现实浏览器中选择 Simulation | Block Parameters 命令，VR Sink 模块参数对话框会再次出现。

(5) 在 Output 选项组中选中 Open VRML viewer automatically 复选框。

这个选项使得在运行模型时，虚拟世界浏览器会自动打开。

(6) 在 Description 文本栏中输入 model18to01。

(7) 单击 Apply 按钮。

(8) 在右侧的树形浏览框中，分别打开 S，B1 和 B2 节点，分别选择 scale，translation 和 translation 选项，为了能够相应的从 Simulink 模型中接收信号，单击 OK 按钮关闭对话框。在 VR Sink 模块上会出现相应的输入端口。

(9) 将连接到 Scope 模块的信号线相应的连接到 VR Sink 模块上，如图 18.29 所示。

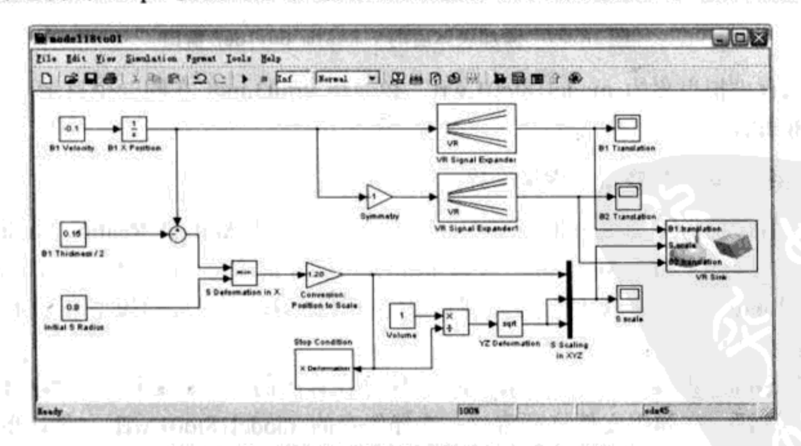


图 18.29 连接相应的信号线到 VR Sink 模块上

(10) 双击 VR Sink 模块，虚拟世界浏览器将会打开。

(11) 在虚拟世界浏览器上方单击【运行】按钮，浏览器中物体变化情况如图 18.30 所示。

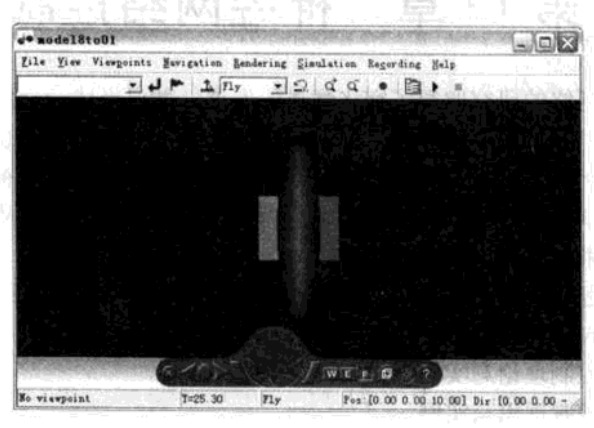


图 18.30　虚拟世界浏览器中的仿真结果

一个简单但是完整的实例介绍完了，Virtual Reality ToolBox 就简单的介绍到这儿，希望能够起到抛砖引玉的作用。

18.8　小　　结

从这个简单小球变形的例子可以了解如何创建一个虚拟环境，用同样的方法可以创建更加复杂的模型。

在 MATLAB 命令窗口中输入：

```
>> vrdemos
```

可以看到更多更加复杂，更加具有现实感的实例。

第 19 章　神经网络控制

近年来，神经网络的研究取得了突飞猛进的发展，并已经成功地应用到了系统的辨识和动态系统的控制当中。本章详细介绍了 Neural Network Blockset 模块库。并介绍了如何利用 MATLAB 命令来生成神经网络的 Simulink 模型。最后通过两个神经网络控制实例介绍了控制模块的基本使用方法，以及在实际应用中模块的设置方法。从仿真的结果可知道取得了非常好的效果，这也为我们进一步应用于实际提供了可行性。

本章主要内容包括：

- Neural Network Blockset 模块库
- 模型参考控制理论与实例
- 模型预测控制理论与实例

19.1 Neural Network Blockset 模块库

神经网络工具箱提供了一套可以在 Simulink 环境中建立神经网络模型的模块。在 MATLAB 工作空间中建立的网络，可以使用 gensim 函数来生成一个相应的 Simulink 模型。

19.1.1 模块介绍

神经网络工具箱的库窗口如图 19.1 所示。从图中可以看出，神经网络工具箱包括 4 个模块库。其中 Control Systems 模块库是在 MATLAB6.5 才增加的。

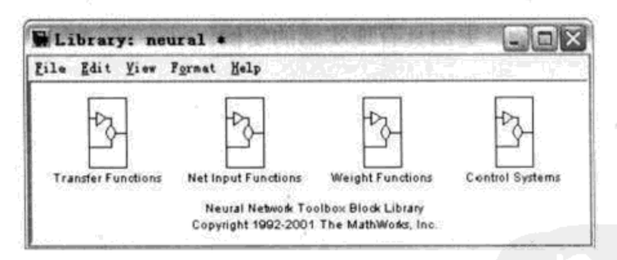


图 19.1　神经网络库

下面分别介绍图 19.1 中 4 个模块库。

- 传递函数模块库 Transfer Functions

传递函数模块库如图 19.2 所示。每个模块都是一种激励函数，能够接受一个网络输入向量，并且相应地产生一个输出向量。

- 网络输入模块库 Net Input Functions

网络输入模块库如图 19.3 所示。每个模块都能够接受任意多个网络输入向量，这些量包括输入向量、加权层输出向量，以及阈值向量，并且相应的产生一个输出向量。

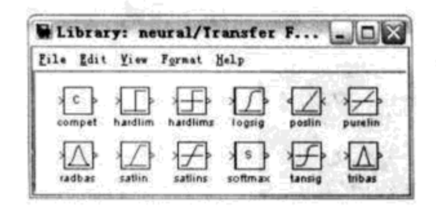


图 19.2　传递函数模块库

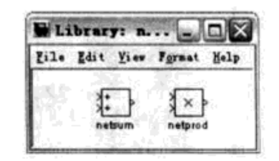


图 19.3　网络输入模块库

这两个模块和 Simulink 中的 Math operations 模块库中的 sum 和 product 模块相似。

- 权值模块库 Weight Functions

权值模块库如图 19.4 所示。权值模块库中的每个模块都以一个神经元中间权向量作为输入，并将其与一个输入向量进行运算，得到神经元的加权输入。

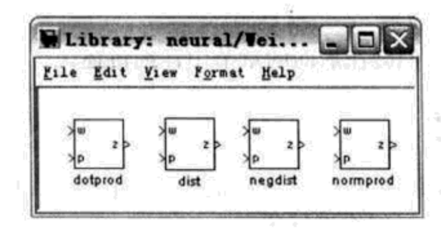


图 19.4　权值模块库

◆ dotprod 模块对应 MATLAB 中的 dotprod 函数，是一个点乘权值函数。
语法：

```
Z = dotprod(W,P)
```

其中 W 为 S×R 权矩阵，P 为 R×Q 矩阵，Q 为列向量。
例如：

```
W = rand(4,3);
P = rand(3,1);
Z = dotprod(W,P)
```

◆ dist 模块对应 MATLAB 中的 dist 函数，计算欧几里得距离权函数。

```
Z = dist (W,P)
```

其中 W 为 S×R 权矩阵，P 为 R×Q 矩阵，Q 为列向量。
例如：

```
W = rand(4,3);
P = rand(3,1);
Z = dist(W,P)
```

◆ negdist 模块和 normprod 模块分别对应 MATLAB 中的 negdist 函数和 normprod 函数，具体用法类似 dotprod 函数和 dist 函数，用 help negdist 和 help normprod 命令查看。

- 控制系统模块库 Control Systems

控制系统模块库如图 19.5 所示，包括以下几个模块组及模块：

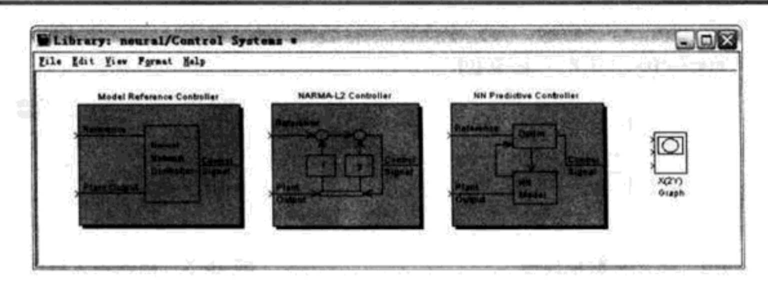


图 19.5 控制系统模块库

- Model reference Controller 模块：模型参考结合神经网络控制。
- NARMA-L2 Controller 模块：反馈线性化结合神经网络控制。
- NN Predictive Controller 模块：神经网络预测控制。
- X(2Y)相图示波器。

前面三个控制模块，我们会在后面的章节中详细讲述。

19.1.2 模块的生成

在 MATLAB 命令空间使用 gensim 函数能够对一个网络生成其模块化描述，这就使得能够很方便地在 Simulink 中对网络进行仿真：

```
gensim(net,st)
```

其中：net 表示神经网络；st 表示样本时间，默认为 1。假如 net 没有输入或者相关层的延迟，即 net.numInputDelays 和 net.numLayerDelays 均为 0，那么可以使用设定 st 为-1，来得到一个连续取样网络。

通过一个简单的例子还说明此函数的用法，来测试 y=3*x，操作步骤如下：

(1) 定义一个输入 p 和相应的目标 t。

```
>> p=[1 2 3 4 5 6 7];
>> t=[3 6 9 12 15 18 21];
```

(2) 利用函数 newlind 来设计神经网络。

```
>> net=newlind(p,t)
net =
    Neural Network object:
    architecture:
         numInputs: 1
         numLayers: 1
       biasConnect: [1]
      inputConnect: [1]
      layerConnect: [0]
     outputConnect: [1]
     targetConnect: [1]
        numOutputs: 1  (read-only)
        numTargets: 1  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)
    subobject structures:
            inputs: {1x1 cell} of inputs
            layers: {1x1 cell} of layers
```



```
        outputs: {1x1 cell} containing 1 output
        targets: {1x1 cell} containing 1 target
         biases: {1x1 cell} containing 1 bias
   inputWeights: {1x1 cell} containing 1 input weight
   layerWeights: {1x1 cell} containing no layer weights
    functions:
         adaptFcn: (none)
          initFcn: (none)
       performFcn: (none)
         trainFcn: (none)
    parameters:
       adaptParam: (none)
        initParam: (none)
     performParam: (none)
       trainParam: (none)
    weight and bias values:
               IW: {1x1 cell} containing 1 input weight matrix
               LW: {1x1 cell} containing no layer weight matrices
                b: {1x1 cell} containing 1 bias vector
    other:
         userdata: (user stuff)
```

(3) 输入测试数据。

```
>> test=[0.5 1.5 2.5 3.5 4.5 5.5 6.5];
```

(4) 使用 sim 函数来测试网络，可以看出神经网络已经正确地解决了问题。

```
>> y=sim(net,test)
y =
   1.5000    4.5000    7.5000   10.5000   13.5000   16.5000   19.5000
```

(5) 调用 gensim 函数生成 net 网络的 Simulink 模型。

```
>> gensim(net,-1)
```

其中-1 表示将生成一个连续采样网络模块。

生成的 Simulink 模型如图 19.6 所示，模型中包括一个线性网络的仿真，这个线性网络输入端连接一个采样输入，输出端连接一个示波器。

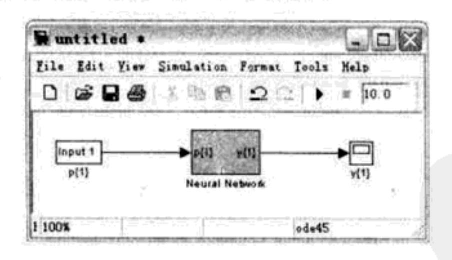


图 19.6　生成神经网络的 Simulink 模型

(6) 双击 Neural Network 模块。在图 19.6 中，并不能看到网络结构信息，打开 Nerual Network 模块，就能够看到 Nerual Network 子系统的构成，如图 19.7 所示。

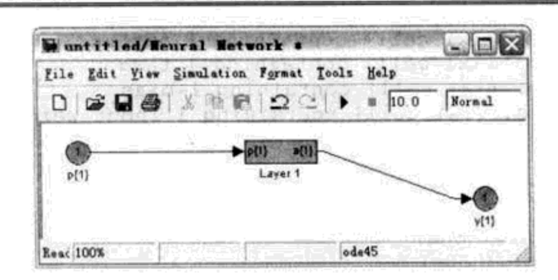


图 19.7 线性神经网络的构成

(7) 双击模块 Layer1，会得到更加详细的信息，如图 19.8 所示。

(8) 双击 b{1}模块就会弹出参数对话框，如图 19.9 所示。双击图 19.8 中 Delays 1 模块和 IW{1,1}模块，仍然会弹出模型窗口，用户可以一直单击下去，直到弹出参数对话框。

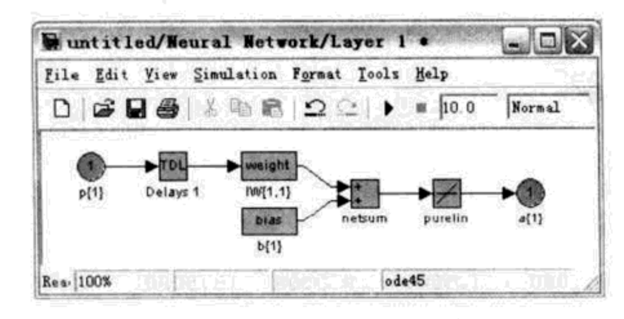


图 19.8 子系统 Layer1 的构成

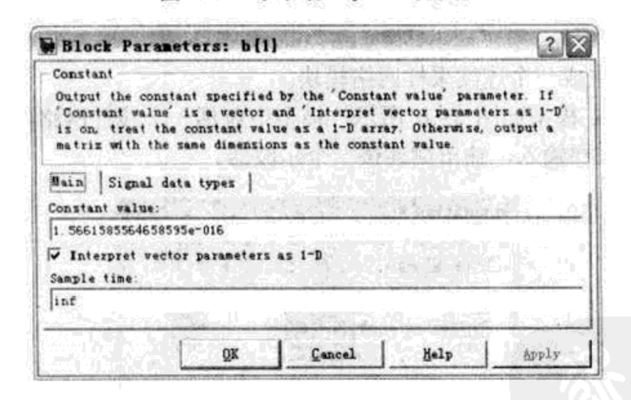


图 19.9 b{1}模块参数对话框

(9) 双击神经网络 Simulink 模型中的{1}模块，在弹出对话框的 Constant value 栏中输入 2.3，如图 19.10 所示，结果如图 19.11 所示为 6.9。保存模型为 model19to01.mdl。

(10) 修改原来的神经网络模型，将 p{1}模块换成 Sine Wave 模块，如图 19.12 所示。运行结果如图 19.13 所示，结果满足 y=3*x 模型。

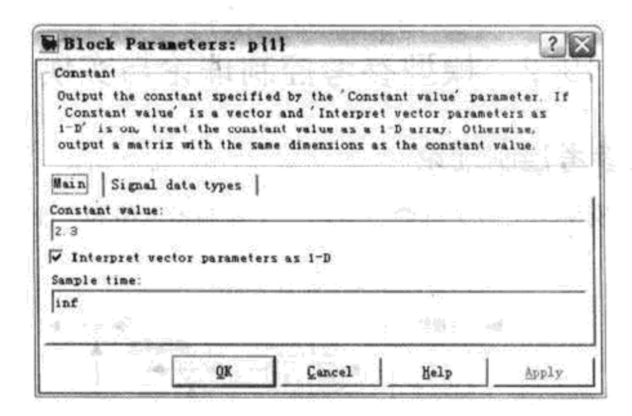


图 19.10　输出结果

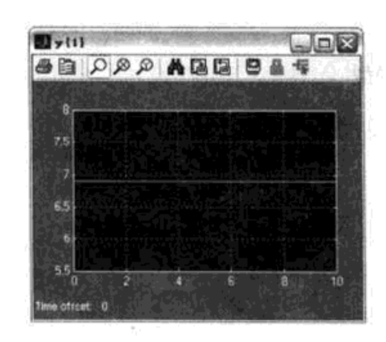


图 19.11　输出结果

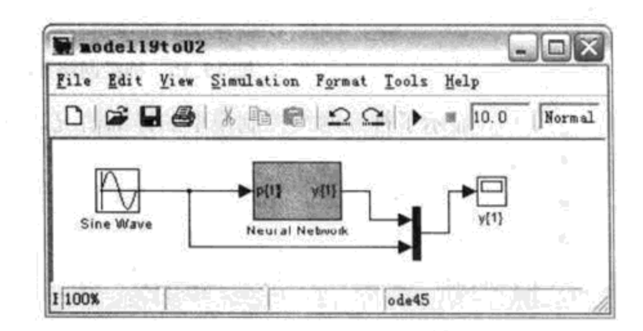


图 19.12　修改后的 Simulink 模型

用户可以重新建立网络，这次离散采样时间为 0.2，具体过程与前面过程类似，可以采用如下命令：

```
>> gensim(net,0.4)
```

然后同样修改模型如图 19.12 所示，保存文件名为 model19to03.mdl，此模型与 model19to02.mdl 中的 Neural Network 模块不同，前者为离散模型，后者为连续模型。仿真结果如图 19.14 所示。

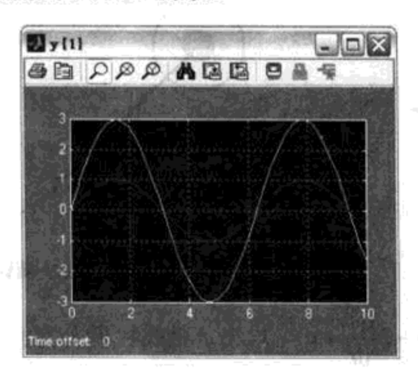


图 19.13　修改模型后的结果

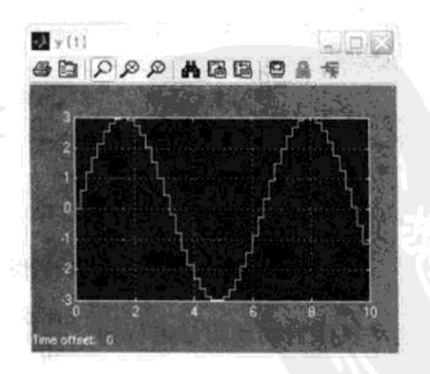


图 19.14　离散模型仿真结果

19.2　模型参考控制理论与实例

19.2.1　模型参考控制理论

神经网络参考模型采用两个神经网络，一个控制器网络和一个实验模型网络，如图 19.15 所示。

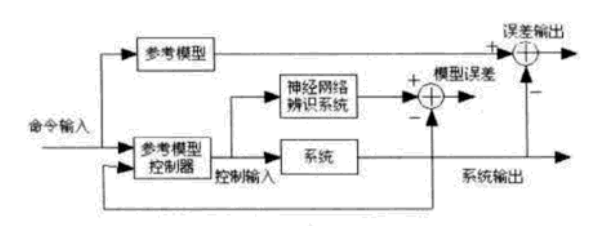


图 19.15　神经网络参考模型

图 19.16 显示了神经网络控制模型的详细情况，有 3 类控制器：

- 延迟参考输入
- 延迟控制输出
- 延迟辨识输出

关于控制器，延迟的数目可以任意选择。

19.2.2　模型参考控制实例分析

1. 建立数学模型

控制一个简单的单连接机械臂，机械臂如图 19.17 所示。

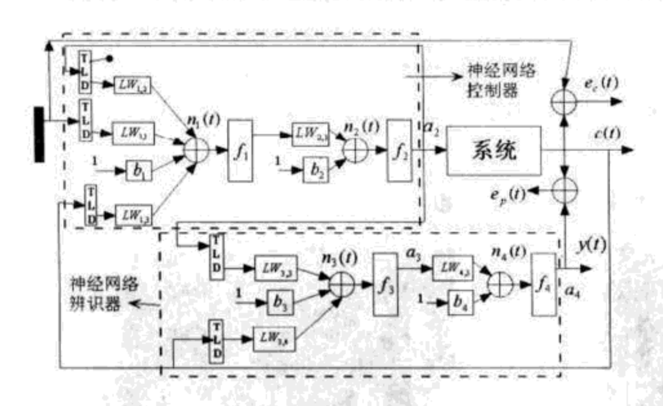


图 19.16　神经网络控制模型

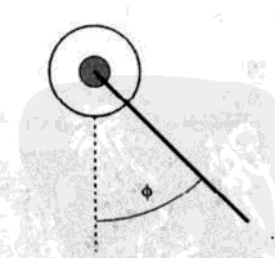


图 19.17　简单的单连接机械臂

运动机械臂的动力学方程为：$\dfrac{\mathrm{d}^2\phi}{\mathrm{d}t^2}=-10\sin\phi-2\dfrac{\mathrm{d}\phi}{\mathrm{d}t}+u$，式中 ϕ 是机械臂与竖直方向的夹角，u 为机械臂的作动力矩。

训练神经网络控制器，让机械臂跟踪如下模型轨迹：$\frac{d^2y_r}{dt^2}=-9y_r-6\frac{dy_r}{dt}+9r$，式中 y_r 为参考模型的输出，r 为参考模型的输入。

2. 建立 Simulink 模型

Neural Network Toolbox4.0.3 提供了这个实例。神经网络使用了 5-13-1 结构，控制器输入包含两个延迟参考输入，两个延迟系统输出和一个延迟控制器输出，采样时间为 0.05 秒。

在 MATLAB 命令窗口输入：

```
>> mrefrobotarm
```

就会自动弹出如图 19.18 所示 Simulink 模型，这是 MATLAB 系统自带的一个实例。在这个模型中，使用到 Model Reference Controller 模块。

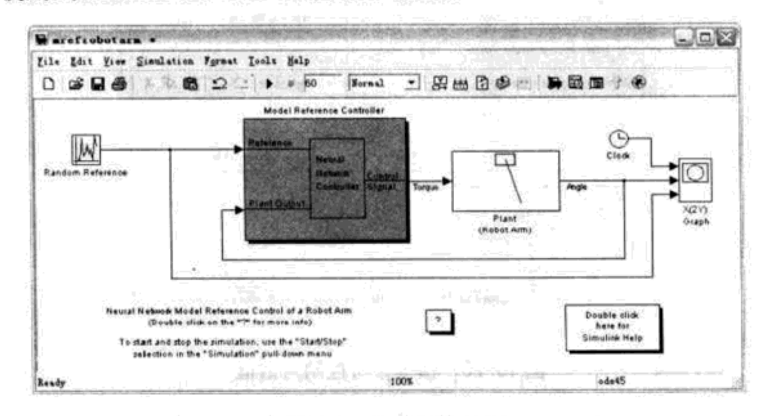


图 19.18　控制系统模型

在图 19.18 中，双击 Plan(Robot Arm)模块，弹出机械臂动力学系统的 Simulink 模型，如图 19.19 所示。对于这个系统的建模，就不在此赘述了，前面的章节有详细的介绍。

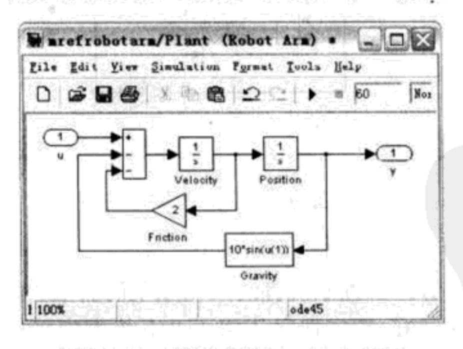


图 19.19　机械臂的 Simulink 模型

控制系统使用了 Model Reference Controller 模块，此模块的输出端(Control Singal)即为扭矩，系统模型的 Angle 输出端连接到 Model Reference Controller 模块的 Plant Output 输入端。参考信号连接到 Model Reference Controller 模块的 Reference 输入端。

3. 模型参考控制对话框

双击 Model Reference Controller 模块，弹出参数对话框，如图 19.20 所示，这个窗口用于训练模型参考控制器。

对于图 19.20 所示窗口，有多项参数可以调整。将光标放到相应的参数名上，就会出现相应的参数说明，如图 19.20 所示。现对参数做一简单的说明。

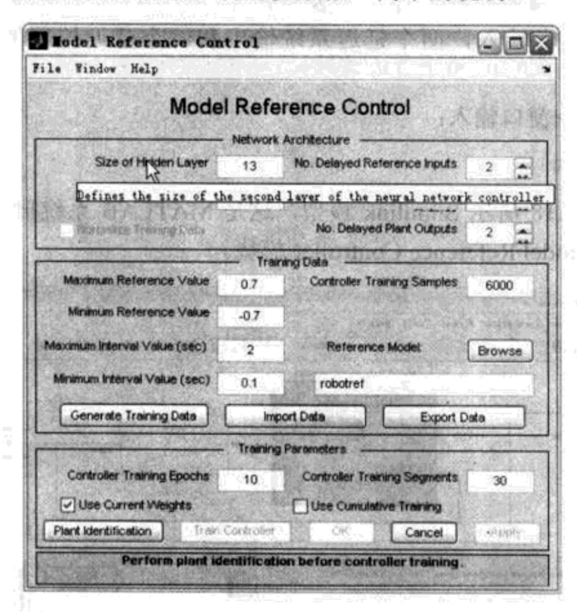


图 19.20　模型参考控制对话框

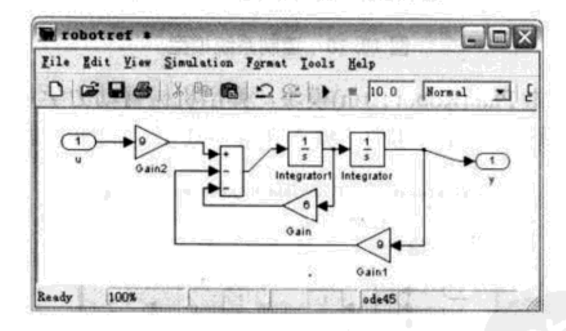


图 19.21　参考模型

参数项说明如下：

- Size of Hidden Layer：设置系统神经网络系统隐含层神经元数目。
- Sampling Interval(sec)：指定程序从 Simulink 模型中采集数据的样本时间。
- Normalize Training Data：指定是否将训练数据标准化。
- No. Delayed Reference Inputs：指定网络的输入延迟。
- No. Delayed Controller Outputs：指定网络的输出延迟。
- No. Delayed Plant Outputs：指定系统的输出延迟。

- Maximum Reference Value(sec)：参考输入的最大值。
- Minimum Reference Value(sec)：参考输入的最小值。
- Maximum Interval Value(sec)：指定一个最大的时间间隔，在此间隔内，随机输入保持不变。
- Minimum Interval Value(sec)：指定一个最小的时间间隔，在此间隔内，随机输入保持不变。
- Controller Training Samples：控制器训练样本数。
- Reference Model：参考模型，本例参考模型如图 19.21 所示。
- Controller Training Epochs：神经网络控制器训练的迭代次数。
- Controller Training Segments：神经网络控制器的训练分批数。
- Use Current Weights：设定是否选择当前的权值用于连续训练。
- Use Cumulative Training：设定是否累计训练。

按钮的用法如下：

- Generate Training Data：生成训练数据。
- Import Data：导入数据。
- Export Data：导出数据。
- Plant Identification：打开系统辨识参数对话框。

4. 系统辨识

单击模型参考控制对话框中的 Plant Identification 按钮，弹出系统辨识参数对话框，如图 19.22 所示。

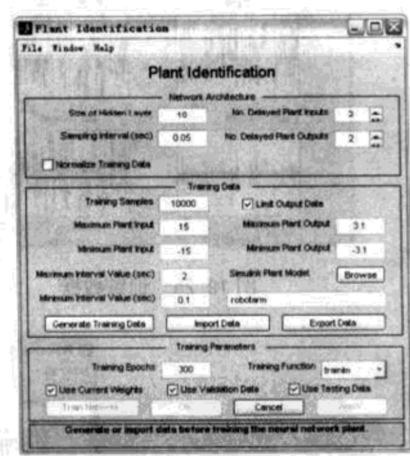


图 19.22　系统辨识窗口

在 Simulink Plant Model 栏中加入受控系统模型，本例为 robotarm，具体模型如图 19.19 所示。可将如图 19.19 所示的模型复制到一个新窗口中，然后保存为 robotarm。因为图 19.19 并不是一个单独的文件，不能够被引用。

这一参数对话框的大多数参数都与神经网络控制器参数对话框类似，而且将鼠标放到相应的参数名上，也会出现相应的参数说明。

参数说明如下：

- No. Delayed Plant Inputs：系统的延迟输入。
- Limit Output Data：是否限制输出数据。
- Maximum Plant Outputs：最大的系统输出。
- Minimum Plant Outputs：最小的系统输出。
- Simulink Plant Model：系统的 Simulink 模型。
- Training Function：神经网络训练方法。

5. 开始训练神经网络

单击系统辨识窗口中的 Generate Training Data 按钮，程序就会产生一系列随机阶跃信号，用来生成数据，如图 19.23 所示。

在图 19.23 中有两个按钮，一个为 Accept Data 按钮，如果单击该按钮，则接受这些数据。另外一个为 Reject Data 按钮，如果单击这个按钮，就不接受这些数据，回到参数对话框，重新产生数据。

单击 Accept Data 按钮后，单击模型辨识窗口中的 Train Network 按钮，辨识神经网络开始训练。训练的效果如图 19.24 所示，表示神经网络的训练结果，包括训练性能，有效性能和测试性能。训练完成后，就会弹出 3 个结果图，如图 10.25～图 10.27 所示，分别表示训练数据，有效数据和测试数据。

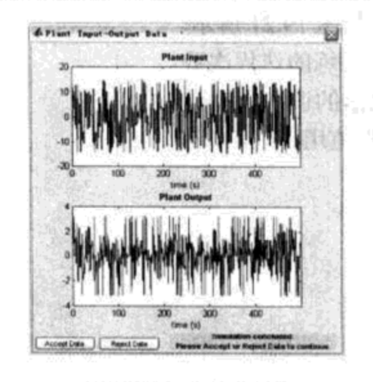


图 19.23　训练数据

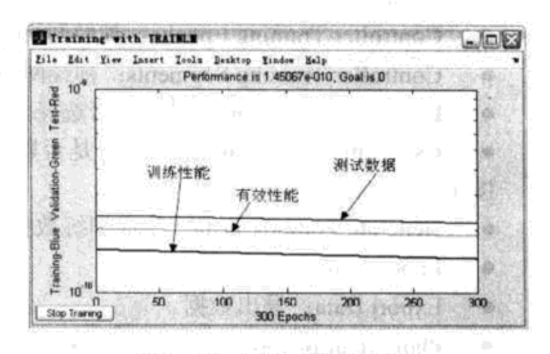


图 19.24　神经网络训练结果

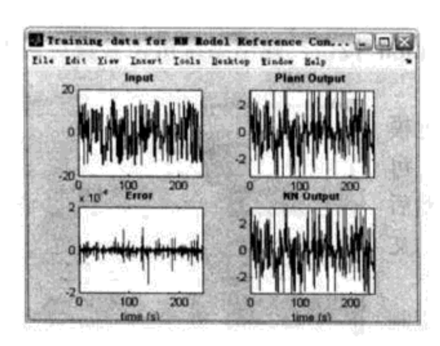


图 19.25　训练数据

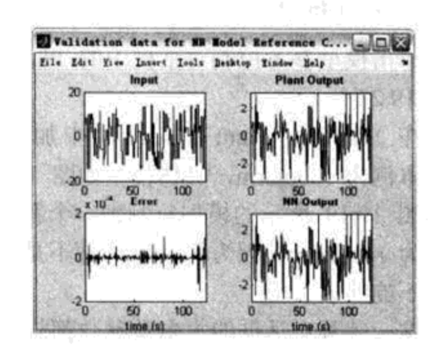


图 19.26　有效数据

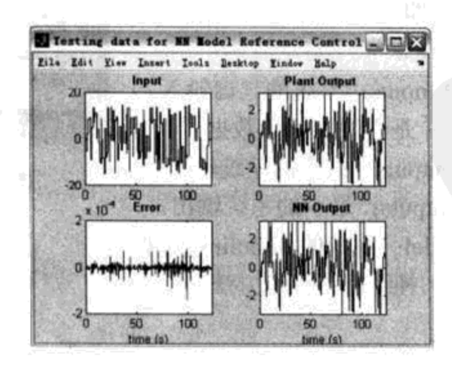


图 19.27　测试数据

6. 训练神经网络控制器

当辨识神经网络训练完成之后，单击 OK 按钮，回到控制器参数对话框。在控制器对话框中单击 Generate Training Data 按钮，产生训练数据，然后在数据窗口单击 Accept data 按钮，如图 10.28 所示。然后回到控制器参数对话框，单击 Train Controller 按钮。完成之后，会弹出控制器训练结果图，从中可以看出跟踪效果非常理想。训练完成之后，单击 OK 按钮，回到 Simulink 模型窗口,将训练好的神经网络控制器权重导入 Simulink 模型中。

7. 系统仿真

运行 Simulink 模型，经过一段时间，就会得到如图 10.29 所示结果。

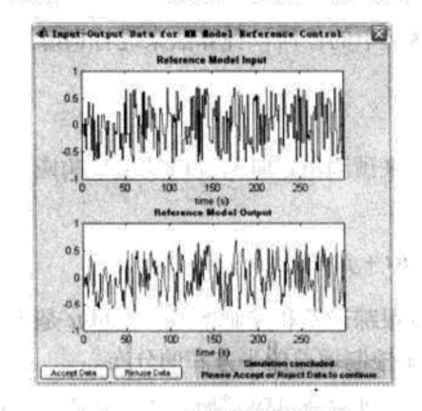


图 19.28　控制器训练数据

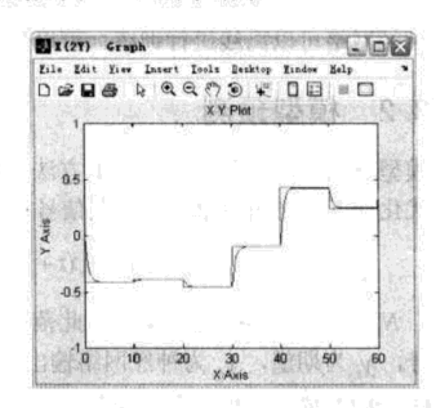


图 19.29　仿真结果

19.3　模型预测控制理论与实例

神经网络预测控制器使用非线性神经网络模型来预测未来模型性能。控制器计算控制输入，以使未来某段时间内的系统性能达到最优。控制的第一步就是得到神经网络辨识模型，然后利用控制器来预测未来神经网络性能。

19.3.1　系统辨识

模型预测控制的第一步就是训练一个神经网络来代替动态系统。预测信号与实际系统误差作为训练信号，具体如图 19.30 所示。

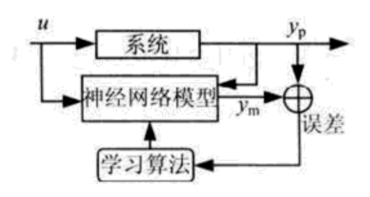


图 19.30　神经网络训练结构图

神经网络模型用到神经网络本身前一时刻的输入和系统前一时刻的输入来预测系统的未来值。神经网络模型结构如图 19.31 所示。

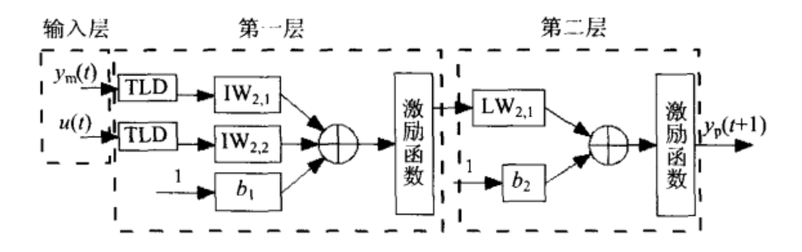


图 19.31　神经网络模型结构

神经网络可以离线进行训练，可以使用神经网络中的任何一种算法来进行训练。

19.3.2　模型预测

模型预测方法是基于水平后退方法，神经网络来预测指定时间段内的模型响应。主要通过优化如下性能函数来确定控制信号。

$$J=\sum_{N_1}^{N_2}(y_r(t+j)-y_m(t+j))^2+\rho\sum_{j=1}^{N_u}(u(t+j-1)-u(t+j-2))^2$$

其中：$N_1,N_2,\cdots,N_u$ 定义范围，在此范围内，计算跟踪误差和控制增益。变量 $\boldsymbol{u}$ 是实验控制信号，y_r 为期望，y_m 为神经网络输出，ρ 反映了控制增益的平方和的分布。

图 19.32 描述了模型预测控制的过程，控制器由神经网络模型和最优化方块组成，通过 J 来确定 $\boldsymbol{u}$。

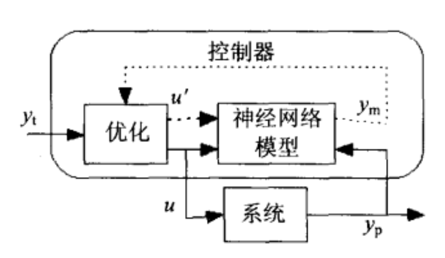


图 19.32　模型预测控制的过程

19.3.3　模型预测控制实例

1. 问题的描述

讨论一个搅拌釜控制的实例模型。搅拌釜模型如图 19.33 所示。

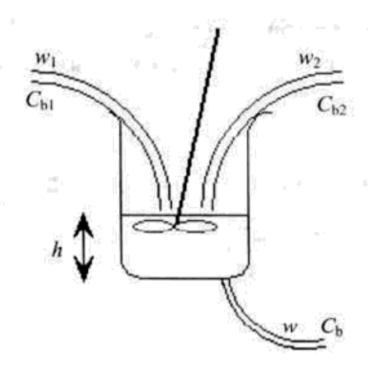


图 19.33　搅拌釜模型

2. 搅拌釜动力学模型

$$\frac{\mathrm{d}h(t)}{\mathrm{d}t}=w_1(t)+w_2(t)-0.2\sqrt{h(t)}$$

$$\frac{\mathrm{d}C_b(t)}{\mathrm{d}t}=(C_{b1}-C_b(t))\frac{w_1(t)}{h(t)}+(C_{b2}-C_b(t))\frac{w_2(t)}{h(t)}-\frac{k_1C_b(t)}{(1+k_2C_b(t))^2}$$

其中 $h(t)$ 为溶液高度，$C_b(t)$ 输出产品浓度，$w_1(t)$ 为浓缩液 $C_{b1}(t)$ 的输入流速，$w_2(t)$ 为浓缩液 $C_{b2}(t)$ 的输入流速。输入常数为 $C_{b1}(t)=24.9$，$C_{b2}(t)=0.1$。消耗常量设置为 $k_1=1$，$k_2=1$。控制目的是通过设置流量来维持稳定浓度。

3. 建立模型

在 MATLAB 命令窗口中输入 predcstr，就会自动弹出系统模型，如图 19.34 所示。用户可以按照系统的模型手动建立 Simulink 模型。在此就不再赘述建模过程了。

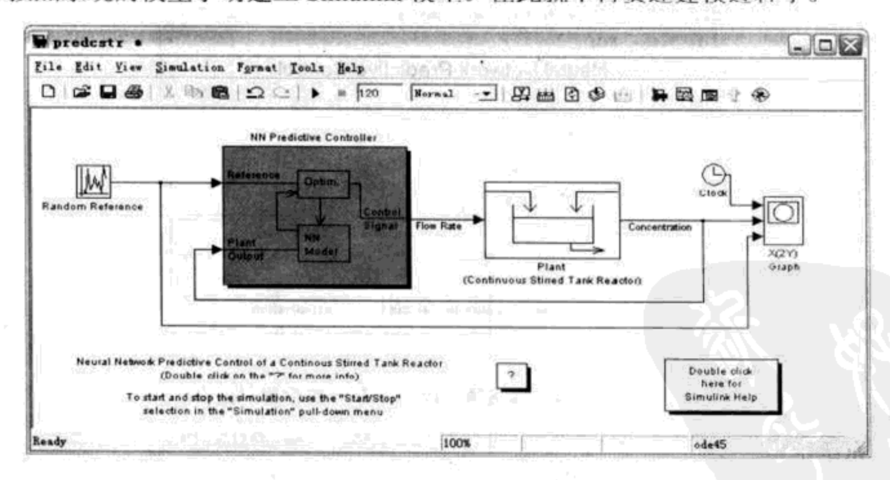


图 19.34　Simulink 模型窗口

双击 Plant(Continuous Stirred Tank Reactor)模块，就会弹出搅拌釜系统的 Simulink 模型，如图 19.35 所示。

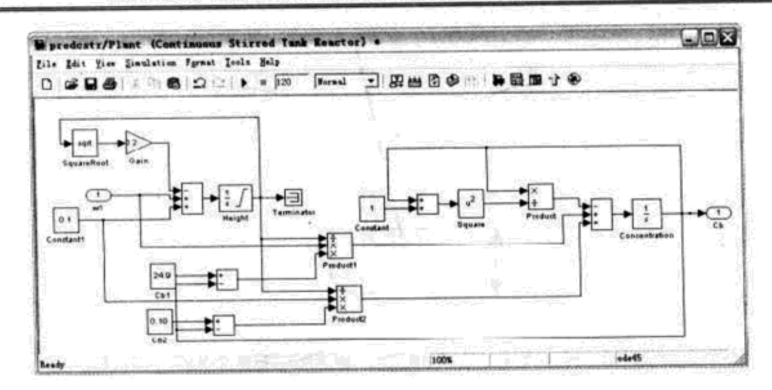


图 19.35 搅拌釜系统的 Simulink 模型

4. 设置控制器参数

双击 NN Predictive Controller 模块，弹出如图 19.36 所示对话框。该对话框用来设计模型预测控制器，设置控制器的相关参数，说明如下：

- Cost Horizon(N2)：设定时间步长，在此步长内，使预测误差达到最小。
- Control Honizon(Nu)：设定时间步长，在此期间控制增量达到最小。
- Minimization Routine：在几个线性搜索程序中确定一个优化程序。
- Control Weight Factor( )：控制权重因子用于对控制中能量的平方和相乘进行加权。
- Search Parameter( )：决定线性搜索何时停止。
- Iterations Per Sample Time：选择每个采样时间中优化算法的迭代次数。

按钮的功能在上一节中介绍过了，在此不赘述。

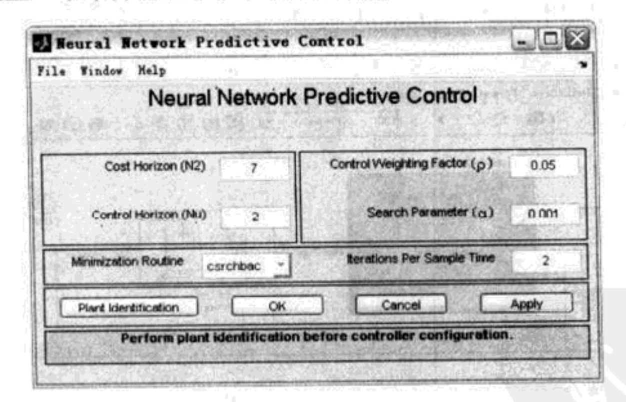


图 19.36 【神经网络预测控制器】参数设置对话框

5. 系统辨识

在【神经网络预测控制器】参数设置对话框中单击 Plant Identification 按钮，将弹出一个对话框，用于设置系统辨识的参数，如图 19.37 所示。其中各种参数设置与上一节类似，用户可以自行设置。

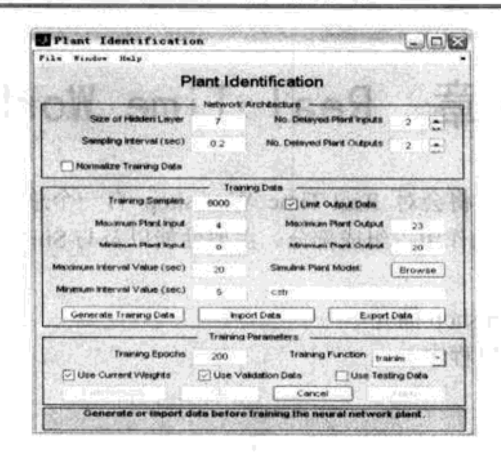


图 19.37　神经网络辨识参数对话框

在使用控制器之前，必须先将辨识神经网络进行训练，用来预测系统的未来输出值。在【神经网络辨识】参数对话框中 Simulink Plant Model 参数 cstr 为搅拌釜的动力学模型，与图 19.36 相同，只不过图 19.36 为子系统模型。可将图 19.36 中系统复制到一个新的 Simulink 模型中，然后保存，获得系统模型 cstr。当然，在此系统已经在文件夹 MATLAB\Toolbox\nnet\nncontrol\中提供了。

辨识的操作步骤如下：

(1) 单击 Generate Training Data 按钮，产生辨识神经网络训练数据，如图 19.38 所示。

(2) 单击 Accept Data 按钮，回到【神经网络辨识】参数对话框。

(3) 单击 Train Networks 按钮进行神经网络训练，训练完成后，会弹出训练结果。

(4) 在【神经网络辨识】对话框中单击 OK 按钮，回到【神经网络预测控制器】设置对话框。

(5) 在【神经网络预测控制器】参数设置对话框中，单击 OK 按钮，完成神经网络控制器设置，回到 Simulink 模型窗口。

(6) 运行 Simulink 模型，仿真结果如图 19.39 所示。

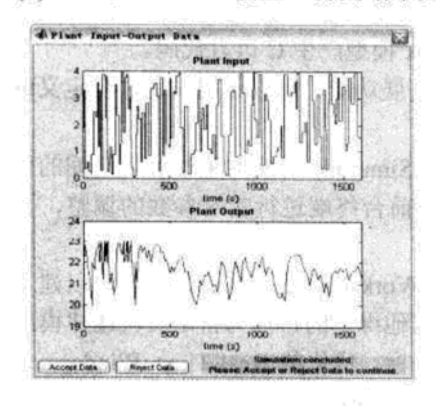


图 19.38　生成训练数据

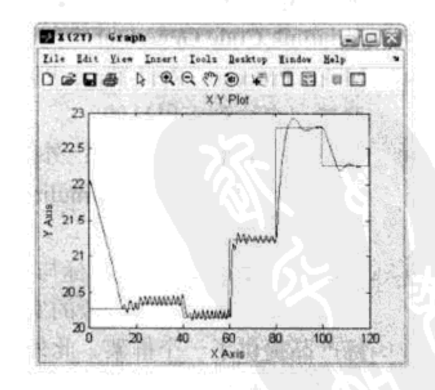


图 19.39　仿真结果

第 20 章　Real-Time Workshop

通过这一章的介绍，将会对 Real-Time Workshop 有一个总体的认识。本章具体阐述了 Real-Time Workshop 的作用、组成部分、主要性能以及与 Simulink 联合建模的方法。

本章主要内容包括：

- Real-Time Workshop 简介
- 生成普通的实时程序
- 产生代码
- 外部模式
- 引用模型代码生成

20.1　Real-Time Workshop 简介

Real-Time Workshop 是 Simulink 和 MATLAB 功能的一种拓展，将从 Simulink 模型中自动产生、打包和编译代码，为不同的系统产生实时应用软件。Real-Time Workshop 是产生代码的基础，提供一个快速生成代码的环境。

结合 MATHWORKS 公司提供的其他工具，Real-Time Workshop 具有如下功能：

- 为不同的目标平台，自动生成代码。
- 从系统设置到执行的快速而直接的方法。
- 与 MATLAB 和 Simulink 实现无缝结合。
- 简单的 GUI。
- 具有良好的开拓性，为用户自行设计打开方便之门。

1. Real-Time Workshop 的主要构成和特性

Real-Time Workshop 的主要构成和特性具体如下：

- Simulink Code Generator：自动为 Simulink 模型产生 C 语言代码。
- Make Process：Real-Time Workshop 的拓展功能，可以让用户通过自定义编译和联接，来达到应用目的。
- Simulink External Mode：外部模式充当 Simulink 与实时测试环境之间的信息桥梁。外部模式在使用 Simulink 时，作为前台终端进行实时参数的调整，数据记录和可视化。
- Targeting Support：将目标与 Real-Time Workshop 捆绑在一起，就可以建立一个实时或者模型环境。一般的实时环境和其他绑定的目标为自定义快速建模或者目标产品提供了一个框架。此外，绑定的目标、优化的 Real-Time Windows Target 和 PC Target 可以让任何一个 PC 变成一个快速样机。
- Rapid Simulations：使用 Simulink Accelerator,S-Function Target，或者 Rapid Simulation Target，可以将仿真速度提高 5～20 倍。利用 Simulink Accelerator，

S-Function Target，或者 Rapid Simulation Target 产生代码是经过高度优化的，为特定模型设定的算法。

- Large-Scale Modeling：支持 Simulink 中的多级建模映射到 Real-Time Workshop 中，从而可以产生多个独立的模型。

2. Real-Time Workshop 第三方编译器

大多数目标创建一个可在工作站上执行的目标。在创建可执行目标之前，Real-Time Workshop 必须和适当的编译器进行连接。下面就介绍如何配置系统，使得 Real-Time Workshop 能够使用编译器。

1) Borland

确保 Borland 环境变量已经被定义，正确地指定 Borland 编译器所在的目录。是否定义了这个 Borland 环境变量，可以在 DOS 窗口输入：

```
set BORLAND
```

如果 Borland 环境变量已经定义了，那么就会返回编译器所在的目录。

如果 Borland 环境变量没有定义，则必须指定安装 Borland 编译器所在的位置。

- 如果系统为 Windows 95 或 Windows 98，那么就将

```
set BORLAND=<编译器的路径>
```

加入到 autoexec.bat 文件中。

- 如果系统为 Windows NT、Windows 2000 或 Windows XP，在系统控制面板中，选择【高级】选项卡，选择【环境】选项，然后将 Borland 的路径添加进去。

用鼠标右击电脑桌面，在弹出的菜单中选择【属性】命令，就会打开系统控制面板。

2) Intel

Real Time Workshop 同样支持 Intel 编译器(version 7.1 应用于 Microsoft Windows)，Intel 编译器需要 Microsoft Visual C/C++ 6.0 或更新的版本。然而，只有 Visual C/C++ 6.0 能够和带有 Intel 编译器的 Real-Time Workshop 联合工作。

在 Real-Time Workshop 生成代码时为了使用 Intel 编译器，要用 mex–setup，然后选择 Intel 编译器。

手动设置 Intel 编译器的操作步骤如下：

(1) 复制文件$(MATLAB)\bin\win32\mexopts\intelc71opts.bat 到期望的目录中，并命名为 mexopts.bat。

(2) 编辑新文件 mexopts.bat，用 Microsoft Visual C/C++ version 6.0 安装根目录替换%MSVCDir%。

(3) 用 Intel Compiler 安装根目录替换%INTELC71% 。

3) LCC

LCC 是自由软件，是 MATLAB 的默认安装。只要利用 MATLAB 自带 LCC 就能和 Real-Time Workshop 共同工作。

4) Microsoft Visual C/C++

利用 MATLAB 命令：

```
mex -setup
```

来定义 Visual C/C++ Versions 5, 6, and 7.1 的使用环境。

在 Windows 系统中，S 函数目标和模型参考仿真目标都是借助 MATLAB 的 mex 命令产生 DLL 文件的。

5) Watcom

注意，Watcom C 编译器不能够再从厂商那里获得了。这个编译器已经由 Open Watcom 组织接管(http://www.openwatcom.org)，为已经使用 Watcom C/C++ 和 Fortran 的用户发布了二进制补丁。Real-Time Workshop 继续自带了 Watcom 相关的目标配置，但是这个政策将会在以后有所改变。

确保 Watcom 环境变量已经被定义，正确地指定 Watcom 编译器所在的目录。是否定义了这个 Watcom 环境变量，可以在 DOS 窗口输入：

```
set WATCOM
```

如果 WATCOM 环境变量已经定义了，那么就会返回编译器所在的目录。如果 WATCOM 环境变量没有定义，必须指定安装 Borland 编译器所在的位置。

- 如果系统为 Windows 95 or Windows 98，那么就将

  ```
  set Watcom =<编译器的路径>
  ```

  加入到 autoexec.bat 文件中。
- 如果系统为 Windows NT、Windows 2000 或 Windows XP，在系统控制面板中，选择【高级】选项卡，选择【环境】选项，然后将 Watcom 的路径添加进去。

如果在设置过程中，遇到"Out-of-Environment Error Message"错误信息，可以用鼠标右击产生这个问题的程序(如 autoexec.bat)，并且从弹出的菜单中选择 Properties 命令，从打开的对话框中选择 Memory 选项卡，并把 Initial Environment 属性设置为所允许的最大值，然后单击 Apply 按钮应用以上设置。

在 Windows 中，Real-Time Workshop 可以使用的编译器版本如表 20.1 所示。

表 20.1 Real-Time Workshop 可以使用的编译器

编译器	版　本
Borland	5.2, 5.3, 5.4, 5.5, 5.6
Intel	7.1
LCC	MATLAB 自带的 LCC
Microsoft Visual C/C++	5.0, 6.0, 7.0
Watcom	10.6, 11.0

在 UNIX 中，Real-Time Workshop 通常使用默认的编译器，cc 编译器可用在几乎所有的操作系统中，除了 SunOS，在 SunOS 中 gcc 是默认的。

3. 编译器优化设置

在极少数情况下，由于编译器的缺点，在 Real-Time Workshop 中应用编译器优化可能会导致生成的可执行程序产生错误的结果，即使代码本身是正确的。

Real-Time Workshop 对每个支持的编译器都应用默认的优化设置。用户通常要遇到如何设置编译器优化问题，例如降低优化设置或关闭优化设置。

20.2 生成普通的实时程序

这一节演示如何利用 Simulink 模型生成 C 代码和普通的实时程序，程序具有以下特点：

- 可以作为一个单机程序执行，独立于外部的定时和事件；
- 它保存数据在 MATLAB 的 MAT 文件中，作为以后分析的数据；
- 可以在 PC 或者 UNIX 环境中生成。

20.2.1 打开演示程序

在 MATLAB 命令行中输入如下命令：

```
>> rtwdemos
```

将会打开一个窗口，如图 20.1 所示，里面包含若干演示组。双击 Modeling Support 组，将弹出如图 20.2 所示的窗口。然后双击 Simulink for control system design(aerospace example)，就会弹出类似于 20.3 所示的窗口，只是与窗口中的说明不同。

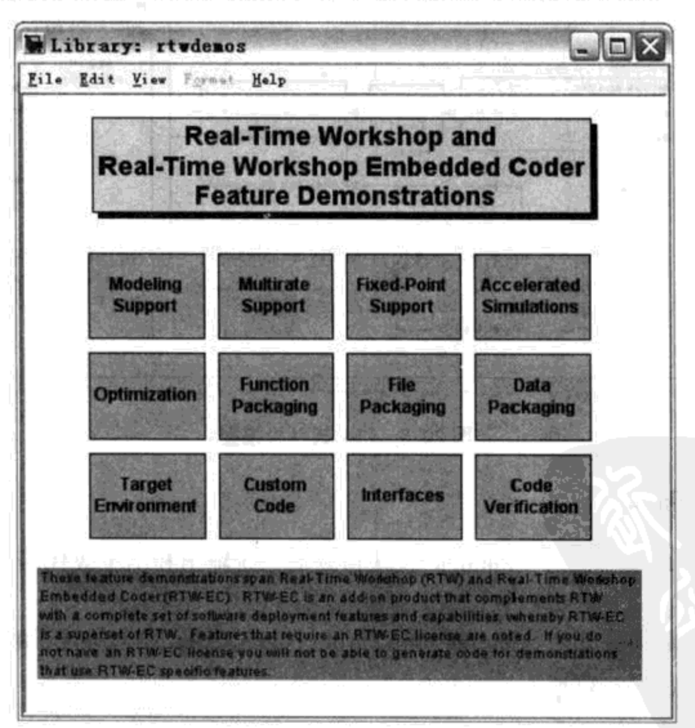


图 20.1 实例演示组

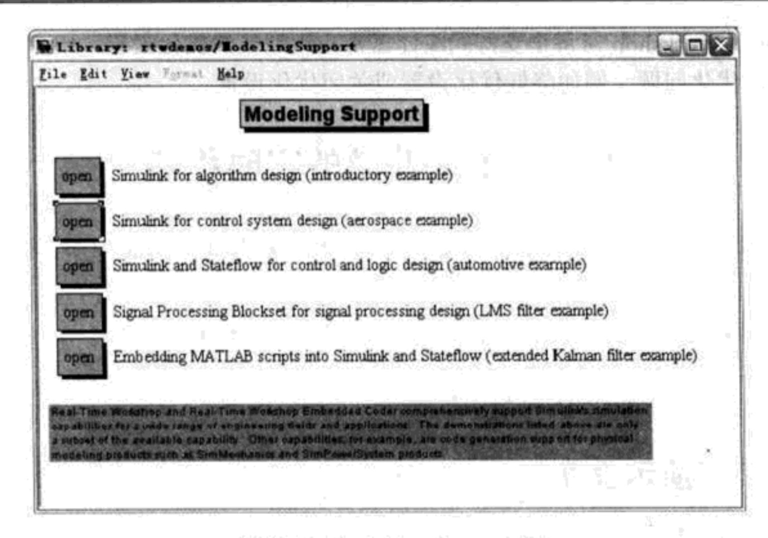


图 20.2　Modeling Support 组

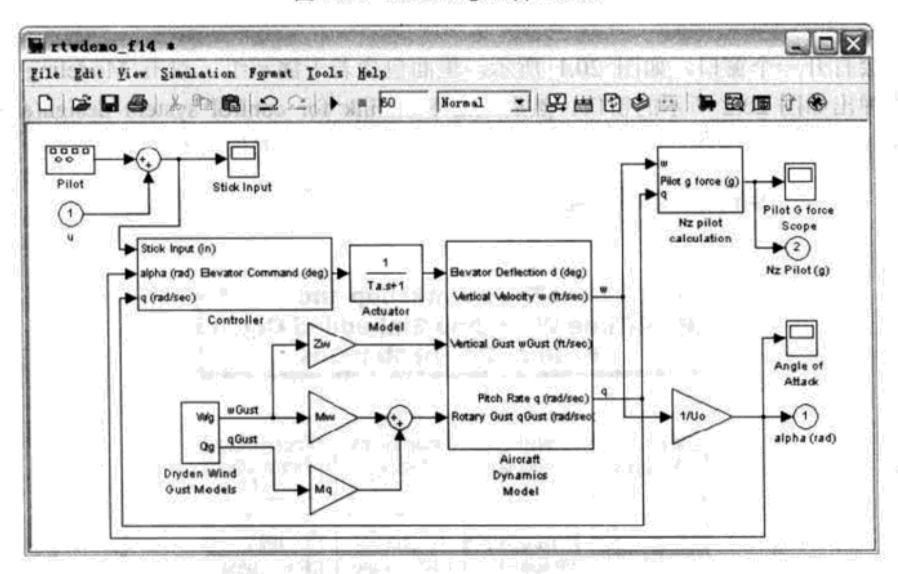


图 20.3　f14 战斗机模型

20.2.2　实例演示

数据保存允许用户将由 Simulink 模块得到的系统输出和由生成代码产生的程序得到的数据进行对比。

这里就利用前面打开的 Simulink for control system design 模型进行讲解。

1. 打开 Simulink 模型

除了前面介绍的方法外，还可以直接在 MATLAB 命令窗口中输入以下命令：

```
>> f14
```

f14 是对一个 f14 飞机纵向运动进行控制的仿真模型。

首先来简单介绍一下 f14 模型。此模型中主要有一个信号发生模块、三个 Scope 模块、四个增益模块和三个子系统模块。信号发生模块用来模拟飞行员的操作，由一个 Scope 模块来显示。另外两个 Scope 模块分别显示系统的输出，分别为飞行员承受的重力和飞行攻角。对于子系统的构造，读者可以双击相应模块打开看看，比较复杂，可能一时难以理解。由于建立模型不是此处的重点，在此就不详细说明。而且生成代码的操作和模型本身的具体内容并没有多大的关系。

2. 生成实时代码

(1) 设置程序参数。

在生成代码之前，必须对一些参数进行设置。没有特别说明，操作都是针对 f14 的。

选择 Simulink 窗口中的 Simulation|Configuration Parameters 命令，打开 Simulink 系统参数对话框，选择 Real-Time Workshop 控制面板，如图 20.4 所示。

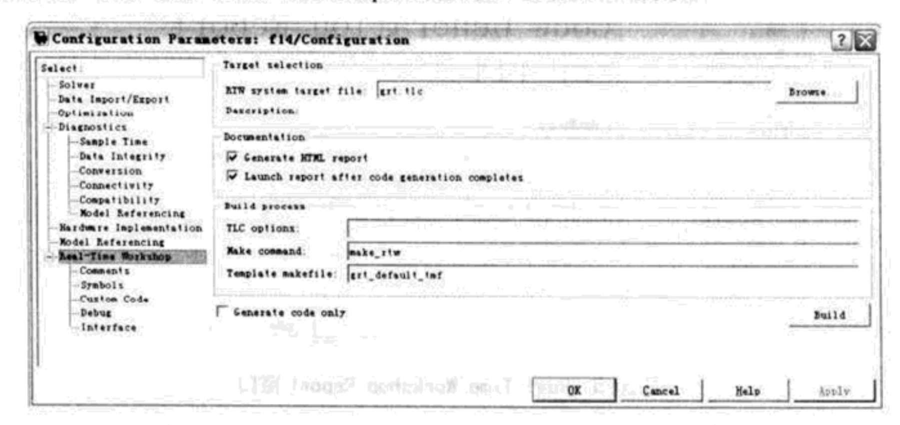


图 20.4 Real-Time Workshop 控制面板

Real-Time Workshop 控制面板提供了若干参数选项，选择模板制作文件(makefile)，产生代码和建立程序的地方。

说明： 对话框中的默认设置不一定和用来产生可实时执行的代码所要求的参数匹配。因此，Simulink 要求用户针对不同的模型来设置不同的参数。如果选中 Generate code only 复选框，表示只产生代码，而不进行生成目标文件和可执行文件。

对于 f14 模型，进行设置的操作步骤如下：

在 Simulink 系统参数对话框的 Solver 控制面板中，设置 Solver options 中的 Type 参数为 Fixed-step；设置 Solver 为 ode5(DormandPrince)求解器；设置 Fixed Step Size 为 0.05；选择 Real-Time Workshop 控制面板，或选择 Tools | Real-Time Workshop | Options 命令打开。

(2) 建立程序。

在 Real-Time Workshop 控制面板中，单击 Build 按钮，就可以生成 C 代码了。在此，

Build 命令调用目标文件 grt.tlc，使用指定的模板文件 rt_default_tmf，来生成 makefile，然后 Real-Time Workshop 利用这个 makerfile 来建立程序。

Real-Time Workshop 的执行过程如下：

① 编译模型，生成 model.rtw 文件，本例为 f14.rtw。

② 调用目标编译器，依次编译 TLC 程序，从 grt.tlc 开始，处理 model.rtw(本例为 f14.rtw)成代码。

③ 从模板 makefile 文件建立一个名为 model.mk(本例为 f14.mk)的 makefile。

④ 如果 Simulink 在模板 makefile 文件指定的宿机上运行，那么实时程序就生成好了，否则，在生成代码后处理就中断，除非用户在模板 makefile 定义和目标相匹配的宿主机。

当 Build 执行完之后，就会弹出一个 Real-Time Workshop Report 窗口，如图 20.5 所示。

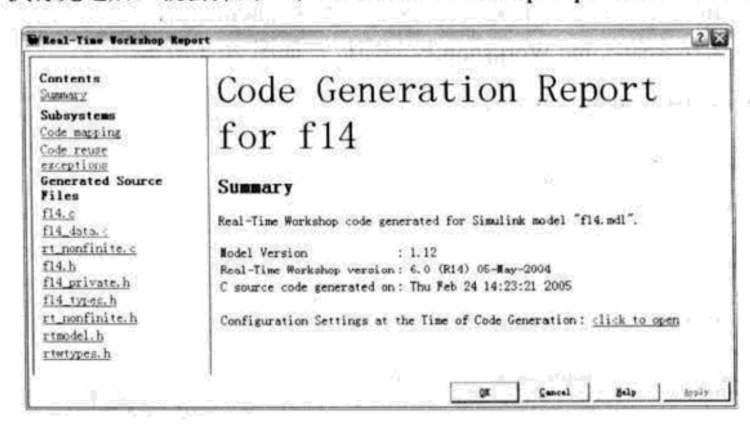


图 20.5　Real-Time Workshop Report 窗口

在弹出窗口的同时，MATLAB 命令窗口会出现相关一系列的信息，这些信息与仿真参数设置相关。部分信息如下：

```
### Starting Real-Time Workshop build procedure for model: f14
### Generating code into build directory: g:\MATLAB7\work\f14_grt_rtw
### Invoking Target Language Compiler on f14.rtw
    tlc
    -r
    g:\MATLAB7\work\f14_grt_rtw\f14.rtw
    G:\MATLAB7\rtw\c\grt\grt.tlc
    -Og:\MATLAB7\work\f14_grt_rtw
    -aReleaseVersion=14.0
    -IG:\MATLAB7\rtw\c\grt
    -Ig:\MATLAB7\work\f14_grt_rtw\tlc
    -IG:\MATLAB7\rtw\c\tlc\mw
    -IG:\MATLAB7\rtw\c\tlc\lib
    -IG:\MATLAB7\rtw\c\tlc\blocks
    -IG:\MATLAB7\rtw\c\tlc\fixpt
    -IG:\MATLAB7\stateflow\c\tlc
    -aEnforceIntegerDowncast=1
略去部分信息
### Writing source file rt_nonfinite.c
### TLC code generation complete.
### Creating HTML report file f14_codegen_rpt.html
```

```
### f14.mk which is generated from G:\MATLAB7\rtw\c\grt\grt_lcc.tmf is
up to date
### Building f14: .\f14.bat
略去部分信息
### Successful completion of Real-Time Workshop build procedure for
model: f1
```

从图 20.5 中可以看出，Real-Time Workshop 自动产生了一系列的文件，单击相应的文件名，图 20.5 所示窗口的右边就会出现相应的代码，在此单击 f14.c 文件，如图 20.6 所示。

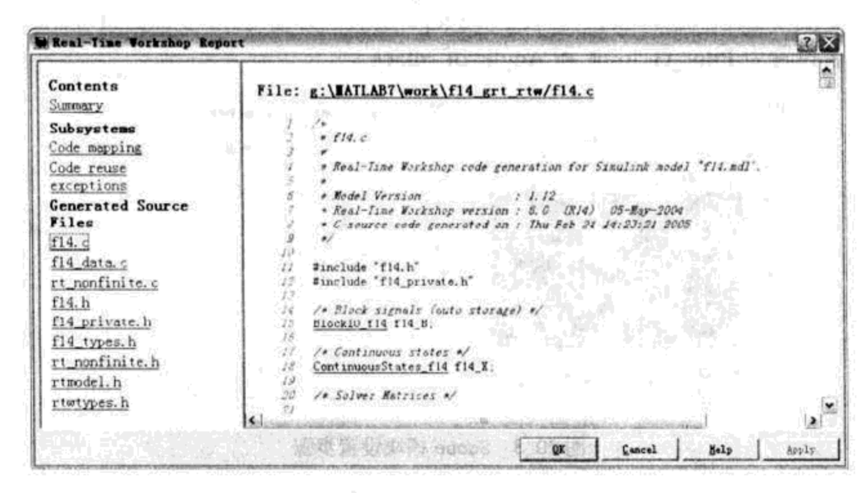


图 20.6 f14.c 代码

(3) 记录数据。

可以使用 MAT-文件来保存系统的状态、输出和时间。只要在 Configurations Parameters 对话框中选择 Workspace Import/Export 控制面板，选择 Time,States 和 Outputs，就可以使 Real-Time Workshop 记录仿真时间、状态和输出。Workspace Import/Export 控制面板主要参数设置部分如图 20.7 所示。

图 20.7 Workspace Import/Export 控制面板

选中 Time,States 和 Outputs 复选框。通过设置 Scope 模块，也可以将相应的数据输入到 MATLAB 的工作空间，然后进行保存。如图 20.8 所示，操作步骤如下：

① 双击要进行数据保存的 Scope 模块。

② 在弹出的 Stick Input 窗口中单击按钮。

③ 在弹出的参数对话框中选择 Data history 面板。

④ 选中 Save data to workspace 复选框，将变量名设置为 Stick_input。

⑤ 单击 OK 按钮。

⑥ 重复步骤①～⑤，设置另外两个 Scope 模块，分别将 Save data to workspace 选项中的变量名设置为 Pilot_G_force 和 Angle_of_attack。

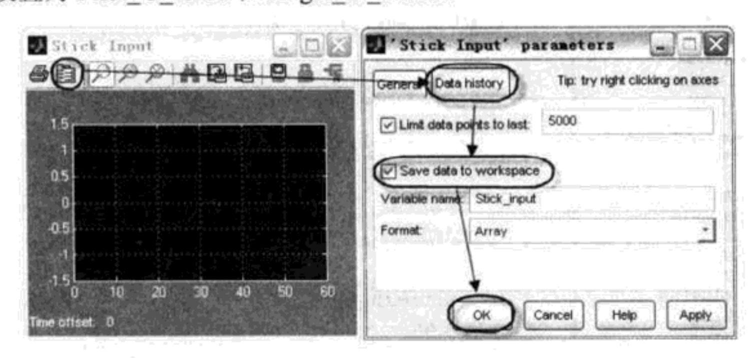


图 20.8 Scope 模块设置步骤

⑦ 在 Real-Time Workshop 控制面板中，单击 Build 按钮，生成可执行代码文件，然后在 MATLAB 命令窗口输入如下的命令：

```
>> !f14
** starting the model **
** created f14.mat **
>> clear
>> load f14
>> who
Your variables are:
rt_Angle_of_attack  rt_Stick_input     rt_xout
rt_Pilot_G_force    rt_tout            rt_yout
```

从中可以看出 f14.mat 包含了可执行程序执行的若干数据变量，由于在 Simulink 中设置了 6 个变量，在 3 个 Scope 模块输出和 3 个 Simulink 系统参数对话框中设置的。这些变量会自动在前面加上一个前缀 rt_，这个前缀读者可以自行设定。

(4) 实时运行接口。

建立实时程序除了根据 Simulink 模型生成代码之外，还需要一系列的其他源文件，这些程序主要有：

- 主程序
- 驱动模块执行代码
- 实现积分算法代码
- 生成 SimStruct 数据结构的代码，SimStruct 用于管理模型的执行

(5) 设置模块 makefile。

有两种 makefile 模板可供使用：

- 在 UNIX 平台中为 grt_unix.tmf 文件。
- 在 PC 平台中为 grt_vc.tmf、grt_wat.tfm 或者 grt_bc.tmf，可根据不同的编译来选择。

这些模板文件在 matlab/rtw/c/grt 目录中。和 S 函数的模板一样，可以将这些文件复制到当前工作目录，然后在模板的基础上进行修改。

模板 grt_vc.tmf 是针对 Visual C/C++用户的，模板 grt_bc.tmf 是针对 Borland C/C++用户的。都必须要求相应的编程环境正确安装和环境变量已经正确定义。

3. 代码验证

完成代码生成之后，就要将代码的计算结果和 Simulink 模型的计算结果进行比较。

(1) 查看 Simulink 模型结果。

① 关闭才打开的 f14 的 Simulink 模型，重新在 MATLAB 命令窗口中输入：

```
>> f14
```

就会将刚开设置好的 f14 模型重新打开。目的是重新获得由于前面使用 clear 命令删除的初始数据。

② 运行仿真。

③ 仿真结果如图 20.9～图 20.11。

④ 在 MATLAB 命令窗口输入：

```
>> who
```

查看数据 Stick_input, Pilot_G_force 和 Angle_of_attack 是否保存到 MATLAB 工作空间当中。

⑤ 可以利用 plot 函数来绘制这些保存数据。

可在 MATLAB 命令窗口输入：

```
>> mfun20to01
```

其中 mfun20to01.m 为绘图文件，内容如下：

```
% mfun20to01.m
figure(1)
plot(Stick_input(:,1),Stick_input(:,2))
xlabel('Time');
ylabel('Stick\_input');
figure(2)
plot(Stick_input(:,1),Pilot_G_force(:,2))
xlabel('Time');
ylabel('Pilot\_G\_force');
figure(3)
plot(Stick_input(:,1),Angle_of_attack(:,2))
xlabel('Time');
ylabel('Angle\_of\_attack');
```



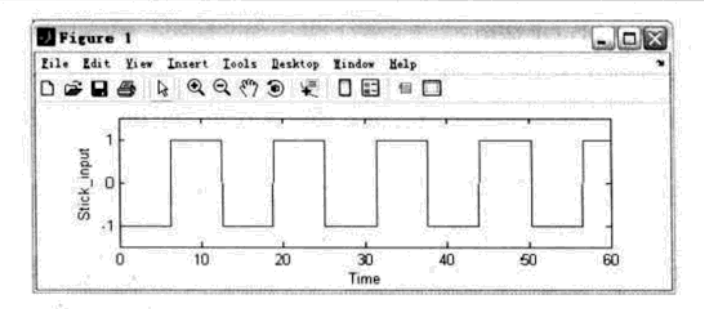


图 20.9 Stick_input 图

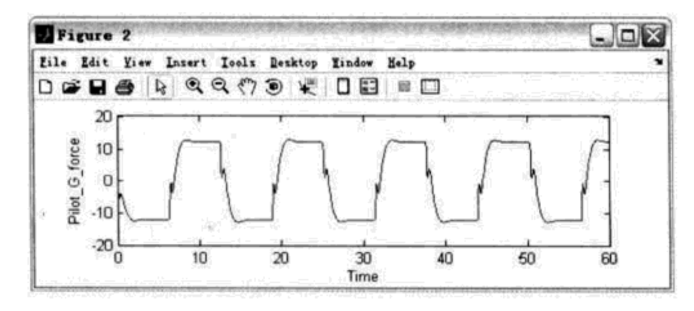


图 20.10 Pilot_G_force 图

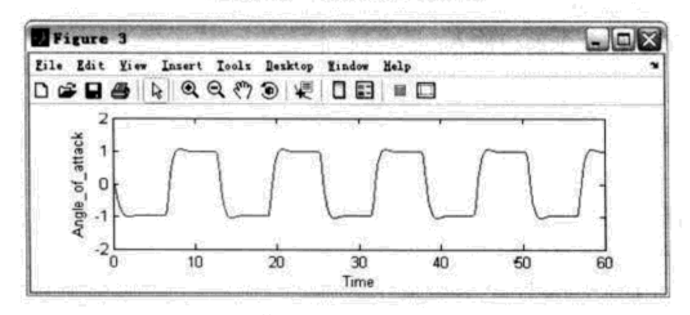


图 20.11 Angle_of_attack 图

(2) 查看生成程序结果。

为了获得有效的数据文件，必须重新运行 f14 可执行文件。

① 设置默认修改变量名参数。打开 Configuration Parameters 参数对话框，选择 Real-Time Workshop configuration 中的 Interface 选项卡。

② 选择 MAT-file variable name modifier 下拉菜单中的 rt_选项，rt_表示在需要保存的每个变量名前面加上一个前缀 rt_。

③ 单击 Apply 按钮。

④ 通过单击图 20.4 中的 Build 按钮产生可执行文件代码。

⑤ 当编译完成，在 MATLAB 命令窗口输入：

```
>> !f14
** starting the model **
** created f14.mat **
>> load f14
```

```
>> whos rt*
  Name                  Size                    Bytes  Class
  rt_Angle_of_attack    1201x2                  19216  double array
  rt_Pilot_G_force      1201x2                  19216  double array
  rt_Stick_input        1201x2                  19216  double array
  rt_tout               1201x1                   9608  double array
  rt_xout               1201x10                 96080  double array
  rt_yout               1201x2                  19216  double array
Grand total is 22819 elements using 182552 bytes
```

⑥ 在 MATLAB 空间中绘制可执行程序执行结果。在 MATLAB 命令空间输入：

```
>> mfun20to02
```

mfun20to02.m 文件内容如下：

```
% mfun20to02.m
figure(1)
plot(rt_Stick_input(:,1),rt_Stick_input(:,2))
xlabel('Time');
ylabel('Stick\_input');
figure(2)
plot(rt_Stick_input(:,1),rt_Pilot_G_force(:,2))
xlabel('Time');
ylabel('Pilot\_G\_force');
figure(3)
plot(rt_Stick_input(:,1),rt_Angle_of_attack(:,2))
xlabel('Time');
ylabel('Angle\_of\_attack');
```

⑦ 结果如图 20.12～图 20.14 所示。

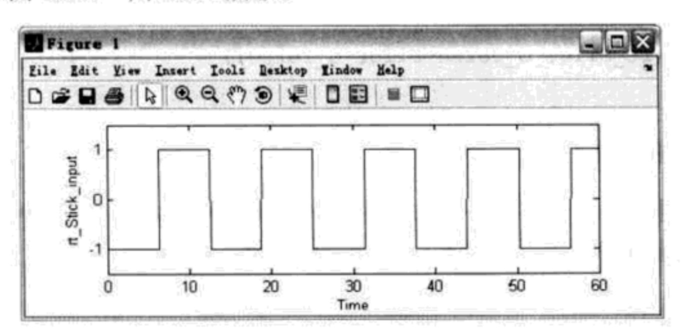


图 20.12 rt_Stick_input 图

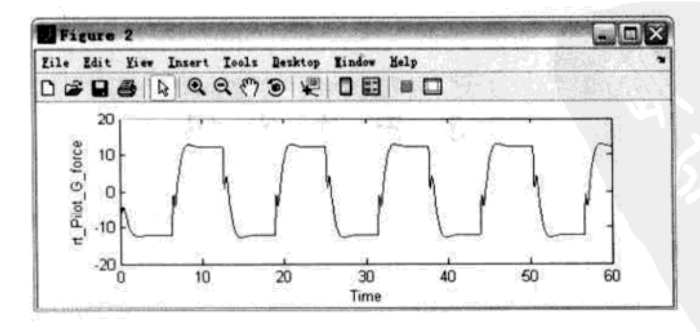


图 20.13 rt_Pilot_G_force 图

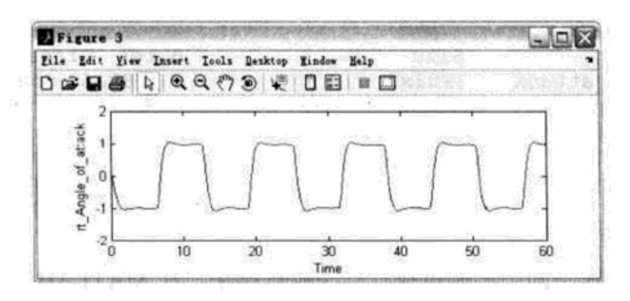


图 20.14　rt_Angle_of_attack 图

(3) Simulink 模型结果与实时程序结果比较。

从上面绘制的结果可以发现，通过两种方法得到的结果几乎一样。

由于已经得到了 Simulink 运行结果和可执行程序的结果，比较两种方法的结果就非常简单了。

比较 Angle_of_attack (Simulink 输出)和 rt_Angle_of_attack (可执行文件输出)，在 MATLAB 命令空间输入如下代码：

```
>> max(abs(rt_Angle_of_attack-Angle_of_attack))
ans =
     0     0
```

可以看出结果在默认精度下完全一致。

比较 Pilot_G_force (Simulink 输出)和 rt_ Pilot_G_force (可执行文件输出)，在 MATLAB 命令空间输入如下代码：

```
>> max(abs(rt_Pilot_G_force-Pilot_G_force))
ans =
     0     0
```

可以看出结果在默认精度下完全一致。

说明： 通常比较结果不会全是 0，在此只是真实地反应本人计算机的运行结果，一般会有一个小小的误差，如 10~15 量级。

误差的导致原因：

- 不同的变异优化
- 语句的顺序
- 运行库

20.3 产生代码

本节将通过一个简单的模型来产生代码，然后再观察 Real-Time Workshop 提供的一些优化性能。操作步骤如下：

(1) 假设建立的模型如图 20.15 所示。

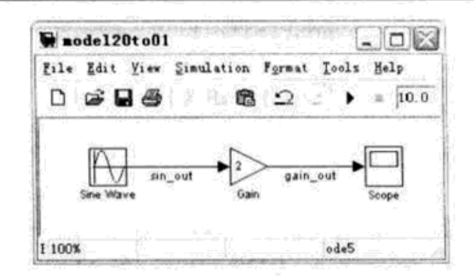


图 20.15　Simulink 模型

(2) 模型设置。

设置模型基本参数的步骤如下：

① 新建一个目录 model20to01，并使之成为当前目录。

```
>>!mkdir model20to01
>>cd realtime
```

② 创建一个 Simulink 模型，并保存为 model20to01.mdl。

③ 拖放相应的 Sine Wave 模块、Gain 模块和 Scope 模块组建模型如图 20.15 所示。

④ 在 Simulink 窗口中，选择 Simulation| Configuration Parameters 命令。

⑤ 在 Solver 控制面板中的 Solver options 区域设置 Type 为 Fixed-step，设置 Solver 为 ode5 (Dormand-Prince)，其他设置为默认，单击 Apply 按钮。

⑥ 打开 Data Import/Export 控制面板，使得所有选项都为非选状态，单击 Apply 按钮。

⑦ 打开 Real-Time Workshop 控制面板，选中 Generate code only 复选框。

选中 Generate code only 复选框表示只产生代码，而不对代码进行编译和连接。因为在此仅研究产生的代码，所以只需要产生代码就足够了。当选中 Generate code only 复选框，右边的 Build 按钮就会变成 Generate code 按钮。确以 System target file 框中为 grt.tlc。

⑧ 单击 OK 按钮。

⑨ 保存模型。

(3) 不用缓冲优化生成代码。

当模块的输入/输出优化性能开启，在任何可能的情况下，Real-Time Workshop 都会使用缓存来保存模块输出数据。在这个练习中，非选此优化功能：

① 打开 Configuration Parameters 对话框中的 Optimization 控制面板，非选 Signal storage reuse 选项和 Implement logic signal as Boolean data(VS.double)选项。

② 单击 Apply 按钮。

③ 打开 Real-Time Workshop 控制面板，单击 Generate code 按钮。

④ 由于选中 Generate code only 复选框，所以只产生代码，而不进行编译和连接。在 MATLAB 命令窗口会产生如下代码：

```
### Starting Real-Time Workshop build procedure for model: model20to01
### Generating code into build directory: g:\MATLAB7\work\ model20to01\
model20to01_grt_rtw
###中间省略若干信息###
```

```
### Writing header file rt_nonfinite.h
### Writing source file rt_nonfinite.c
### TLC code generation complete.
### Creating project marker file: rtw_proj.tmw
### Creating model20to01.mk from G:\MATLAB7\rtw\c\grt\grt_lcc.tmf
### Successful completion of Real-Time Workshop build procedure for
model: model20to01
```

⑤ 产生的代码在系统自动创建的目录 model20to01_grt_rtw 中，包含了模型的计算结果的文件为 model20to01_grt_rtw\ model20to01.c，在 MATLAB 编辑器中打开这个文件。在 MATLAB 命令窗口中输入：

```
>> edit model20to01_grt_rtw\model20to01.c
```

⑥ 在文件 model20to01.c 中，函数 model20to01_output 在文件的开头部分。

生成的 C 代码由若干个程序组成，这些程序执行 Simulink 模块所定义的算法。执行引擎按适当的时间顺序执行这些代码。

在 model20to01 代码中，函数 model20to01_output 实际执行的是一个带有增益的正弦波输出，在每一个时间步都要调用。

由于没有使用缓冲优化，model20to01_output 分配一个缓冲给每一个模块输出。这些缓冲(rtB.sin_out, rtB.gain_out)是全局模块 I/O 数据结构，调用代码 model20to01_B，具体表达式如下：

```
/* Block signals (auto storage) */
BlockIO_realtime01 model20to01_B;
BlockIO_ model20to01 数据在 model20to01.h 中定义如下:
/* Block signals (auto storage) */
typedef struct _BlockIO_realtime01 {
  real_T sin_out;                    /* '<Root>/Sine Wave' */
  real_T gain_out;                   /* '<Root>/Gain' */
} BlockIO_ model20to01;
输出代码如下:
/* Model output function */
static void realtime01_output(int_T tid)
{
  /* Sin Block: '<Root>/Sine Wave' */
  realtime01_B.sin_out = realtime01_P.SineWave_Amp *
    sin(realtime01_P.SineWave_Freq * realtime01_M->Timing.t[0] +
    realtime01_P.SineWave_Phase) + realtime01_P.SineWave_Bias;
  /* Gain: '<Root>/Gain' */
  realtime01_B.gain_out = realtime01_B.sin_out * realtime01_P.Gain_Gain;
}
```

⑦ 在 GRT 目标中，model20to01_output 被一个封装函数 MdlOutputs 调用。可以在 model20to01.c 文件尾部找到这个 MdlOutputs 函数，形式如下：

```
void MdlOutputs(int_T tid) {
  realtime01_output(tid);
}
```

(4) 不用缓冲优化生成代码。

① 打开 Optimization 控制面板，选中 Signal storage reuse 复选框，而不选中 Implement logic signal as boolean data(VS.double)复选框。

② 此时 Code generation 选项组 Enable local block outputs, Reuse block outputs 和

通过这个命令，就对 vdptop 模型转换，并将转换结果存放到当前目录。并会在 MATLAB 命令窗口显示一些转换信息，如果转换成功，就会得到：

```
ans =
     1
```

当转换成工作之后，在当前工作目录中就会出现以下文件：

- model20to03.mdl：你所修改的 vdp 模型。
- model20to03_converted.mdl：由 M 文件创建的顶级模型，里面包含一个模型模块。
- vdpmult.mdl：子系统 vdpmult 创建的模型。
- vdpmult_msf.dll：Windows 静态库文件，当 model20to03 模型调用 vdpmult 子系统时被执行。
- /slprj：为生成模型引用代码建立的目录。

(2) 在 MATLAB 命令窗口输入：

```
>> model20to03_converted
```

打开顶级模型文件。打开示波器，运行模型进行仿真。模型就会调用 vdpmult_msf 目标进行仿真，模型如图 18.28 所示，结果如图 18.29 所示。

引用模型目标代码放在 slprj/sim 子目录中。GRT, ERT 以及其他 Real-Time Workshop 目标生成的代码都放在各自的单独命名的子目录中。

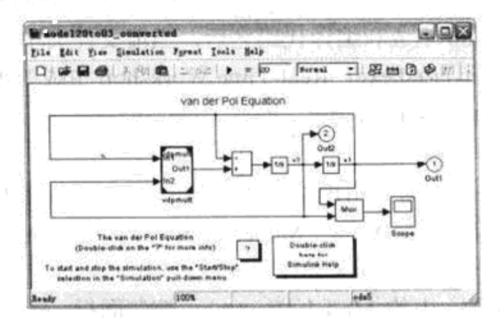


图 20.28 model20to03_converted 模型

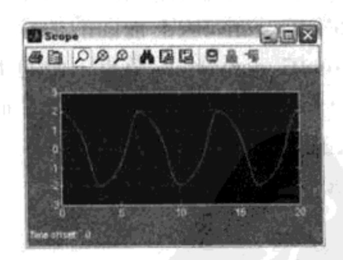


图 20.29 model20to03_converted 模型运行结果

3. 生成 GRT 目标的引用模型代码

sl_convert_to_model_reference 函数只用于 vdpmult 子系统的模型创建和目标文件的生

成。这一部分，就为子系统对应的模型和顶级模型生成代码，并运行这个可执行代码：

(1) 确定 MATLAB 当前工作目录是 model20to03。

(2) 打开 model20to03_converted。

(3) 在 Simulink 窗口中选择 Simulation|Configuration Parameters 命令，或者使用 Ctrl+E 组合键。则打开 model20to03_converted 的 Configuration Parameters 参数对话框。

(4) 打开 Data Import/Export 控制面板，在 Save to workspace 选项组中选中 Time 和 Output 复选框，然后单击 Apply 按钮。

这就可以使得 model20to03_converted 模型在进行仿真时，记录运行的时间和数据。

(5) 为顶级模型和应用模型生成 GRT 代码和可执行文件。可通过以下方式之一：

- 打开 configuration parameter 参数对话框，在 Real-Time Workshop 控制面板中单击 Build 按钮。
- 在 Simulink 窗口中，选择 Tools| Real-Time Workshop|Build Model 命令。
- 使用 Ctrl+B 组合键。

在执行过程中，MATLAB 窗口会出现相关信息，如下所示：

```
### Checking the status of model reference SIM target for model vdpmult
used in 'model20to03_converted'
### Model reference SIM target for vdpmult is up to date
###省略若干内容…###
*** Created executable: model20to03_converted.exe
### Successful completion of Real-Time Workshop build procedure for
model: model20to03_converted
```

在 Real-Time Workshop 生成和编译代码之后，当前文件夹中多了两个新文件。

- model20to03_converted.exe：Real-Time Workshop 生成的可执行文件。
- model20to03_converted_grt_rtw：Real-Time Workshop 建立的目录，包含顶级模型生成的代码。

Real-Time Workshop 同样为引用模型生成代码，不过是放在 slprj 目录中。

(6) 在 Simulink 窗口中选择 View | Model Explorer 命令，利用 Model Explorer 查看新建立的 model20to03_converted_grt_rtw 目录。

(7) 在 Model Explorer 中的 Model Hierarchy 控制面板，展开 model20to03_converted 模型节点。

(8) 单击 Code for model20to03_converted 前面的加号，展开节点。

(9) 单击 Code for model20to03_converted 节点下面的 Top Model 项目。

一系列 model20to03_converted 生成代码就会出现在 Contents 面板中，具体如下：

- model20to03_converted.c
- model20to03_converted_data.c
- rtmodel.h
- model20to03_converted_types.h
- model20to03_converted.h
- model20to03_converted_private.h
- model20to03_converted.mk

可以在 Contents 面板单击相应的文件，然后在右边的框中查看代码，如图 20.30 所示。

(10) 查看 vdpmult.mdl 模型。在 Model Explorer 中的 Model Hierarchy 面板中，右击标有 vdpmult (vdpmult)的模块图标(最后一个选项)，在弹出的菜单中选择 Open Model 'vdpmult'命令，打开模型如图 18.31 所示。

(11) 查看 vdpmult.mdl 模型的代码，在 Model Explorer 中的 Model Hierarchy 面板中,单击标有 vdpmult 选项，在 Model Explorer 窗口右边就会显示引用模型的文件(在/slprj/grt directory 中)，如图 20.32 所示。

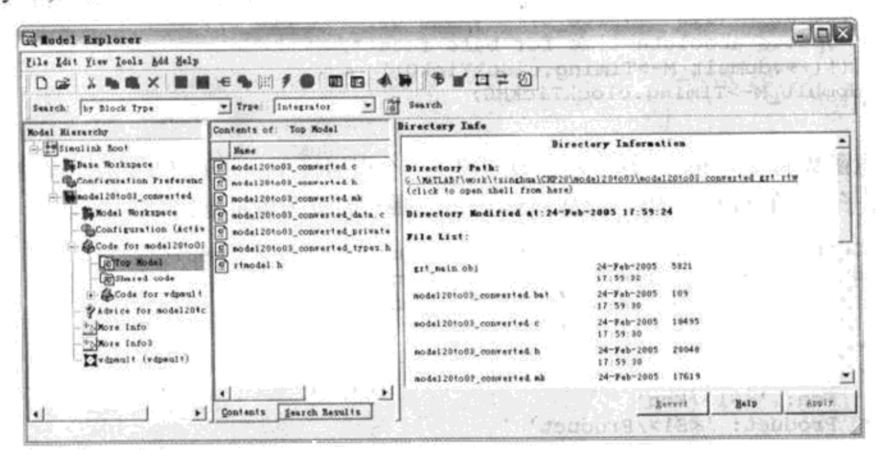


图 20.30　Model Explorer

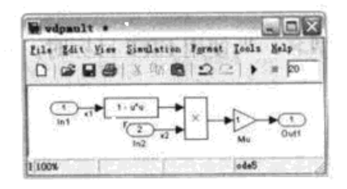


图 20.31　vdpmult 模块

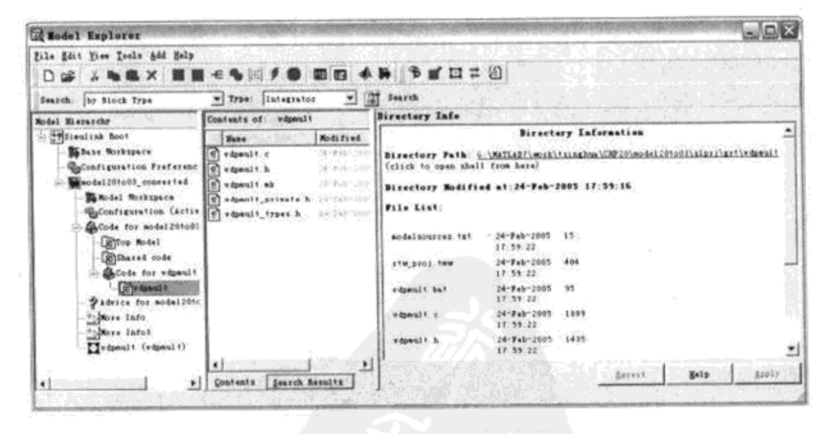


图 20.32　引用模型的文件

可以在 Contents 面板中单击其中的 vdpmult.c 文件，然后在右边的框中查看代码。其中 vdpmult 输出代码如下：

```
/* Output and update for referenced model: 'vdpmult'*/
void mr_vdpmult(const real_T *rtu_0, const real_T *rtu_1,
            real_T *rty_0, RT_MODEL_vdpmult *vdpmult_M)
{
  /* local block i/o variables */
  /* Gain: '<Root>/Mu' incorporates:
   * Fcn: '<Root>/Fcn'
   * Product: '<Root>/Product'
   */
  (*rty_0) = (1.0 - (*rtu_0) * (*rtu_0)) * (*rtu_1);
  /* Update absolute time for base rate */
  if(!(++vdpmult_M->Timing.clockTick0))
++vdpmult_M->Timing.clockTickH0;
}
```

注意函数的前缀 mr_，这表示函数来源于一个引用模型。此处是 model20to03 模型中 vdpmult 子系统的 GRT 代码。RTW system code 属性设置为可重复使用的函数。

```
/* Output and update for atomic system: '<Root>/vdpmult' */
void model20to03_vdpmult(real_T rtu_In1, real_T rtu_In2,
                rtB_model20to03_vdpmult *localB)
{
  /* local block i/o variables */
  /* Gain: '<S1>/Mu' incorporates:
   * Fcn: '<S1>/Fcn'
   * Product: '<S1>/Product'
   */
  localB->Mu = (1.0 - rtu_In1 * rtu_In1) * rtu_In2;
}
```

可重复使用的函数前缀为模型名，当使用 ERT 目标时，可以修改这个标示符的结构。



参 考 文 献

1. 张志涌. 精通 MATLAB6.5 版. 北京：北京航空航天大学出版社，2003
2. 薛定宇，陈阳泉. 基于 MATLAB/Simulink 的系统仿真技术与应用，北京：清华大学出版社，2002
3. 祝效华，廖伟志，黄永安. CAD/CAE/CFD/VPT/SC 软件协作技术，北京：中国水利水电出版社，2004

推荐网络资源

1. 动力学与控制技术论坛，http://www.dytrol.com
2. 中国仿真互动，http://www.simwe.com
3. 中国科研网，http://www.SciEi.com
4. 岩学论坛，http://bbs.matwav.com
5. 水木清华数学工具版，http://www.smth.org
6. http://www.mathtools.net
7. http://www.mathworks.com
8. http://www.geatbx.com/